

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of :
Takuji KAWAMOTO :
Serial No. NEW : **Attn: APPLICATION BRANCH**
Filed March 18, 2004 : Attorney Docket No. 2004-0445A

DATA MEMORY CACHE UNIT AND DATA
MEMORY CACHE SYSTEM

CLAIM OF PRIORITY UNDER 35 USC 119

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

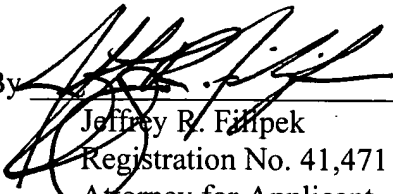
Sir:

Applicant in the above-entitled application hereby claims the date of priority under the International Convention of Japanese Patent Application No. 2003-078026, filed March 20, 2003, as acknowledged in the Declaration of this application.

A certified copy of said Japanese Patent Application is submitted herewith.

Respectfully submitted,

Takuji KAWAMOTO

By 
Jeffrey R. Filipek
Registration No. 41,471
Attorney for Applicant

JRF/krq
Washington, D.C. 20006-1021
Telephone (202) 721-8200
Facsimile (202) 721-8250
March 18, 2004

THE COMMISSIONER IS AUTHORIZED
TO CHARGE ANY DEFICIENCY IN THE
FEES FOR THIS PAPER TO DEPOSIT
ACCOUNT NO. 23-0975



日 本 国 特 許 庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出 願 年 月 日 2 0 0 3 年 3 月 2 0 日
Date of Application:

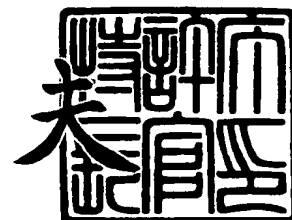
出 願 番 号 特 願 2 0 0 3 - 0 7 8 0 2 6
Application Number:
[ST. 10/C]: [J P 2 0 0 3 - 0 7 8 0 2 6]

出 願 人 松 下 電 器 産 業 株 式 会 社
Applicant(s):

2 0 0 3 年 1 0 月 2 9 日

特許庁長官
Commissioner,
Japan Patent Office

今 井 康



【書類名】 特許願

【整理番号】 2968150011

【提出日】 平成15年 3月20日

【あて先】 特許庁長官殿

【国際特許分類】 G06F 12/08

【発明者】

【住所又は居所】 愛知県名古屋市中区栄2丁目6番1号白川ビル別館5階
株式会社松下電器情報システム名古屋研究所内

【氏名】 川本 琢二

【特許出願人】

【識別番号】 000005821

【氏名又は名称】 松下電器産業株式会社

【代理人】

【識別番号】 100097445

【弁理士】

【氏名又は名称】 岩橋 文雄

【選任した代理人】

【識別番号】 100103355

【弁理士】

【氏名又は名称】 坂口 智康

【選任した代理人】

【識別番号】 100109667

【弁理士】

【氏名又は名称】 内藤 浩樹

【手数料の表示】

【予納台帳番号】 011305

【納付金額】 21,000円

【提出物件の目録】

【物件名】 明細書 1

【物件名】 図面 1

【物件名】 要約書 1

【包括委任状番号】 9809938

【書類名】 明細書

【発明の名称】 データメモリキャッシュ制御装置及びデータメモリキャッシュ制御方法

【特許請求の範囲】

【請求項1】 CPUがアクセスするメモリ領域のアクセスするアドレスの連続性に関する情報とアクセスするアドレスの方向性に関する情報に基づいてキャッシュメモリの制御を行う、

データメモリキャッシュ制御装置。

【請求項2】 CPUからの連続したアドレスのメモリ領域に対する連続書き込み時にキャッシュミスが発生した時、次に書き込みが行われるアドレス方向のメモリ領域に対しては主記憶装置からキャッシュメモリへの読み込みを行わない、データキャッシュメモリ制御装置。

【請求項3】 CPUによる連続したアドレスのメモリ領域に対する連続読み込み時にキャッシュミスが発生した時、既に読み込みが行われたアドレス方向のメモリ領域に対してはキャッシュメモリから主記憶装置への書き込みを行わない、データキャッシュメモリ制御装置。

【請求項4】 CPUからの連続したアドレスのメモリ領域に対する連続書き込み時に、既に書き込みが行われたアドレス方向のメモリ領域で、現在書き込みが行われているメモリ領域とのアドレス距離が一定以上離れたメモリ領域に対して、バスの空き時間を使って、予めキャッシュメモリから主記憶装置への書き込みを行う、データキャッシュメモリ制御装置。

【請求項5】 CPUによる連続したアドレスのメモリ領域に対する連続読み込み時に、今後読み込みが行われるアドレス方向のメモリ領域で、現在読み込みが行われているメモリ領域とのアドレス距離が一定以上近いメモリ領域に対して、バスの空き時間を使って、予め主記憶装置からキャッシュメモリへの読み込みを行う、データキャッシュメモリ制御装置。

【請求項6】 CPUがアクセスするメモリ領域のアクセスするアドレスの連続性に関する情報とアクセスするアドレスの方向性に関する情報に基づいてキャッシュメモリの制御を行う、

データメモリキャッシュ制御方法。

【発明の詳細な説明】

【0 0 0 1】

【発明の属する技術分野】

本特許出願に係る発明（以後単に「本発明」とも言う）は、CPUが主記憶装置をアクセスする時、主記憶装置のアクセス速度が遅いことによるオーバーヘッドを回避し、パフォーマンスを向上させることを目的とするメモリキャッシュ（キャッシュメモリとも言う）の制御装置及び制御方法に関するものである。

【0 0 0 2】

特に、データを記憶するデータメモリのキャッシュに関するものであり、更にいわゆるスタックメモリ、即ち、プッシュ・ポップ命令によってアドレスが連続的にアクセス（データの読み込みや書き込み）が行われるデータメモリのキャッシュに関するものである。

【0 0 0 3】

【従来の技術】

従来の、プッシュ・ポップ命令に必要な時間を減少させ、キャッシュ内にプッシュ・ポップ後の不要なデータを保持しないようにするための、スタックキャッシュの制御方式及びスタックキャッシュには、例えば、主記憶装置上のデータ構造として最後に入力したデータが最初に出力されるスタック構造をサポートし、スタックへのデータ入力をプッシュ、前記スタックからのデータ出力をポップとして、他のデータ書込み／読出しと区別してデータ処理を行うマイクロコンピュータを備えたマイクロコンピュータシステムにおいて、前記マイクロコンピュータがプッシュするとき前記主記憶装置に代わって該当アドレスとデータの組を保持し、前記主記憶装置にプッシュされていない保持データを前記マイクロコンピュータが外部にアクセスしていない時を見計らって上記主記憶装置にプッシュし、前記マイクロコンピュータがポップするとき該当データを保持していれば前記主記憶装置に代わって該当データを前記マイクロコンピュータに出力し、前記マイクロコンピュータのポップで不要になったデータを後にポップされる可能性のあるスタック底部のデータに置き換えるため、前記マイクロコンピュータが外部

にアクセスしていない時を見計らって前記主記憶装置からポップすることを特徴とするスタックキャッシュの制御方式があった（例えば、特許文献1参照。）。

【0004】

【特許文献1】

特開平5-143330号公報

【0005】

【発明が解決しようとする課題】

しかしながら前記従来のスタックキャッシュ制御装置及びスタックキャッシュ制御方法では、通常のデータメモリに対するランダムアクセスを前提としたメモリアクセス高速化手法と何等変わることなく、スタックメモリと言う規則性のあるアクセスの特徴を生かすことができず、高速化の効率に問題があった。

【0006】

このことは特に近年ユビキタスコンピューティング等のためにニーズが増大しているJava（R）（TM）システム等、スタックアクセスの比重が大きいメモリシステムでは特に大きな問題となる。

【0007】

具体的な問題点を次に説明する。

【0008】

例えばスタックに対するデータプッシュが、主記憶装置上のメモリアドレスの増加方向に行われたと仮定し、更にこのスタックがキャッシュメモリにキャッシュされていたと仮定する。

【0009】

そしてあるデータAのプッシュによってキャッシュミスが発生したと仮定する。

【0010】

この条件下では、キャッシュミスが発生したメモリアドレスから1キャッシュ単位分の新しいデータメモリの内容が、主記憶装置からキャッシュメモリに読み込まれる。

【0011】

この状態を図 25 に示す。図 25 はアドレス 90 のメモリ領域にスタックポインタ 913 があり、このアドレス 90 のメモリ領域にデータ A がプッシュされた状態を示している。

【0012】

この時、図 25 に示すようにデータ A のプッシュによってスタックポインタ 913 がアドレス 89 のメモリ領域からアドレス 90 のメモリ領域に移動する。

【0013】

そして更に同じ図 25 に示すようにアドレス 90、91、92、93 のメモリ領域が 1 つのキャッシュ単位であり、この 1 つのキャッシュ単位のアドレス 90 のメモリ領域が始めてアクセスされたので、このアドレス 90、91、92、93 のメモリ領域からなる 1 キャッシュ単位が、主記憶装置からキャッシュメモリに読み込まれる。

【0014】

しかしながらこのキャッシュメモリに読み込まれた 1 キャッシュ単位分の新しいデータメモリの内容は、スタックアクセスの規則性に従う限り、続けて行われるデータプッシュによって上書きされる内容であり、決して読み出されることは有り得ないデータである。

【0015】

そしてこのように決して読み出されることの有り得ないデータを、主記憶装置からキャッシュメモリに読み込むことは高速化の効率に大きな弊害であった。

【0016】

逆にスタックに対するデータポップが、主記憶装置上のメモリアドレスの減少方向に行われたと仮定し、更にこのスタックがキャッシュメモリにキャッシュされていたと仮定する。

【0017】

そしてあるデータ B のポップによってキャッシュミスが発生したと仮定する。

【0018】

この条件下では、キャッシュミスが発生したメモリアドレスよりも大きなメモリアドレスから 1 キャッシュ単位分のデータメモリの内容が、キャッシュメモリ

から主記憶装置にライトバックされる。

【0019】

この状態を図26に示す。図26に示すようにデータBのポップによってスタックポインタ923がアドレス90のメモリ領域からアドレス89のメモリ領域に移動する。

【0020】

そして更に同じ図26に示すようにアドレス90、91、92、93のメモリ領域が1つのキャッシュ単位であり、この1つのキャッシュ単位以外のアドレス89のメモリ領域にスタックポインタが移動したので、このアドレス90、91、92、93のメモリ領域からなる1キャッシュ単位が、キャッシュメモリから主記憶装置にライトバックされ、キャッシュメモリが開放される。

【0021】

しかしながらこの主記憶装置にライトバックされた1キャッシュ単位分のデータメモリの内容は、スタックアクセスの規則性に従う限り、既に行われたデータポップによって読み出された内容であり、決して再び読み出されることは有り得ない。そしてこのように決して読み出されることの有り得ないデータを、キャッシュメモリから主記憶装置にライトバックすることは高速化の効率に大きな弊害であった。

【0022】

また、スタックに対するデータプッシュが、主記憶装置上のメモリアドレスの増加方向に行われたと仮定し、更にこのスタックがキャッシュメモリにキャッシュされていたと仮定する。

【0023】

そしてあるデータC、D、・・・、Eのプッシュが連続して実行されたと仮定する。

【0024】

この条件下では、現在スタックに対してプッシュ・ポップを行う領域であるスタックポインタの示すアドレスよりも低いアドレスのメモリ領域に多くのスタックプッシュされたデータ、今の例ではCやDが記憶されており、この低いアドレ

スのメモリ領域に記憶されたデータは、暫くの時間的な間、ポップによって読み出される可能性は非常に低い。

【0025】

この状態を図27に示す。図27はアドレス194のメモリ領域にスタックポインタ933があり、遥か下の例えばアドレス80のメモリ領域からこのアドレス194のメモリ領域まで、連続してC、D、・・・、Eのデータがプッシュされた状態を示している。

【0026】

この時図27に示すように、通常ならばアドレス80からアドレス83までの1つのキャッシュ単位はキャッシュメモリがフルにならない限りキャッシュメモリ領域を占めたままであり、主記憶装置にライトバックされることはない。

【0027】

そしてキャッシュメモリがフルになった時に始めて、主記憶装置にライトバックされ、キャッシュメモリを他のメモリ領域のために明け渡すことになる。

【0028】

このようにキャッシュメモリがゼロになり、新しいキャッシュメモリ領域に対する要求が発生してからライトバックを行うことは高速化の効率に大きな弊害であった。

【0029】

例えば図27に示すように、現在のスタックポインタ933の位置であるアドレス194のメモリ領域よりも遥かに下のメモリ領域、例えばアドレス80～83のメモリ領域は当分の間、ポップによって読み出される可能性は非常に低く、例えばこのようなアドレス80～83のメモリ領域である1つのキャッシュ単位のキャッシュメモリの内容を、予め主記憶装置にライトバックしておくことによって、キャッシュメモリの余裕がなくなった段階で慌ててキャッシュメモリの内容を主記憶装置に書き込み、キャッシュメモリに空き領域を作るよりも、メモリアクセス高速化の効率を大きく改善することができる。

【0030】

更にまた、スタックに対するデータポップが、主記憶装置上のメモリアドレス

の減少方向に行われたと仮定し、更にこのスタックがキャッシュメモリにキャッシュされていたと仮定する。

【0031】

そしてあるデータ F、G、・・・、H のポップが連続して実行されたと仮定する。

【0032】

この条件下では、現在スタックに対してプッシュ・ポップを行う領域であるスタックポインタの示すアドレスよりも低い値のアドレス領域、例えば今の例ではアドレス 84～87 のメモリ領域にスタックプッシュされているデータは、すぐに続けてポップによって読み出される可能性が非常に高い。

【0033】

この状態を図 28 に示す。図 28 はアドレス 91 のメモリ領域にスタックポインタ 943 があり、遥か上の例えばアドレス 196 のメモリ領域からこのアドレス 91 のメモリ領域まで、連続して F、G、・・・、H のデータがポップされた状態を示している。

【0034】

この時図 28 に示すように、通常ならばアドレス 84 からアドレス 87 までの 1 つのキャッシュ単位はこの領域に属する 1 つのメモリ、例えばアドレス 87 のメモリ領域がポップの実行によってアクセスされない限り主記憶装置からキャッシュメモリ領域に読み込まれることはない。

【0035】

そして例えばこのアドレス 87 のメモリ領域がポップの実行によってアクセスされた時に始めて、主記憶装置からキャッシュメモリ上に読み出されることになる。

【0036】

このように新しい 1 つのキャッシュメモリ単位に属する領域が始めてアクセスされ、新しい 1 つのキャッシュメモリ領域に対するデータの要求が発生した後に、主記憶装置からキャッシュメモリ領域にデータの読み込みを行うことは高速化の効率に大きな弊害であった。

【0037】

例えば図28に示すように、現在のスタックポインタ943の位置であるアドレス91のメモリ領域のすぐ下のメモリ領域、例えばアドレス84～87のメモリ領域を占める1つのキャッシュ単位は、すぐに続けてポップによって読み出される可能性が非常に高く、例えばこのようなアドレス84～87のメモリ領域である1つのキャッシュ単位のデータを、予め主記憶装置からキャッシュメモリに読み込んでおくことによって、キャッシュメモリ上にないメモリ領域に対するアクセスが行われた段階で慌てて主記憶装置の内容をキャッシュメモリに読み込むよりも、メモリアクセス高速化の効率を大きく改善することができる。

【0038】**【課題を解決するための手段】**

前記課題を解決するため、本特許出願に係る請求項1に記載の発明は、CPUがアクセスするメモリ領域のアクセスするアドレスの連続性に関する情報とアクセスするアドレスの方向性に関する情報に基づいてキャッシュメモリの制御を行う、データメモリキャッシュ制御装置である。

【0039】

前記課題を解決するため、本特許出願に係る請求項2に記載の発明は、CPUからの連続したアドレスのメモリ領域に対する連続書き込み時にキャッシュミスが発生した時、次に書き込みが行われるアドレス方向のメモリ領域に対しては主記憶装置からキャッシュメモリへの読み込みを行わない、データキャッシュメモリ制御装置である。

【0040】

前記課題を解決するため、本特許出願に係る請求項3に記載の発明は、CPUによる連続したアドレスのメモリ領域に対する連続読み込み時にキャッシュミスが発生した時、既に読み込みが行われたアドレス方向のメモリ領域に対してはキャッシュメモリから主記憶装置への書き込みを行わない、データキャッシュメモリ制御装置である。

【0041】

前記課題を解決するため、本特許出願に係る請求項4に記載の発明は、CPU

からの連続したアドレスのメモリ領域に対する連続書き込み時に、既に書き込みが行われたアドレス方向のメモリ領域で、現在書き込みが行われているメモリ領域とのアドレス距離が一定以上離れたメモリ領域に対して、バスの空き時間を使って、予めキャッシュメモリから主記憶装置への書き込みを行う、データキャッシュメモリ制御装置である。

【0042】

前記課題を解決するため、本特許出願に係る請求項5に記載の発明は、CPUによる連続したアドレスのメモリ領域に対する連続読み込み時に、今後読み込みが行われるアドレス方向のメモリ領域で、現在読み込みが行われているメモリ領域とのアドレス距離が一定以上近いメモリ領域に対して、バスの空き時間を使って、予め主記憶装置からキャッシュメモリへの読み込みを行う、データキャッシュメモリ制御装置である。

【0043】

前記課題を解決するため、本特許出願に係る請求項6に記載の発明は、CPUがアクセスするメモリ領域のアクセスするアドレスの連続性に関する情報とアクセスするアドレスの方向性に関する情報に基づいてキャッシュメモリの制御を行う、データメモリキャッシュ制御方法である。

【0044】

【発明の実施の形態】

本特許出願に係る発明（以後、「本発明」とも言う）の実施の形態について図を参照して説明する。

【0045】

（実施の形態1）

本発明の第1の実施の形態であるデータメモリキャッシュ制御装置（以後、特に区別すべき場合を除いて単に「キャッシュ制御装置」と言う）のブロック構成を図1に示す。命令メモリキャッシュ制御装置については本発明と直接関係しないので説明を省略する。

【0046】

本実施の形態ではキャッシュ制御装置は大きく、演算ユニット113、メモリ

マネジメントユニット 115（以後、「MMU 115」と言うこともある）、スタック対応データキャッシュユニット 117（以後特に区別すべき場合を除いて単に「キャッシュユニット 117」と言うこともある）、転送制御ユニット 119、主記憶装置 131 から構成される。

【0047】

演算ユニット 113 は通常 MPU または（狭義の）CPU またはプロセッサと呼ばれるもので各種の演算命令、データ移動命令、制御命令その他の命令を命令メモリから取り出して実行する。

【0048】

以後、これらの一例として MPU を考え、「MPU 113」と言うが、これらを全て含むものを示している。

【0049】

MPU 113 は特にメモリアドレスを記憶する専用レジスタ SP（スタックポインタ）を有している。このスタックポインタが示すメモリ番地をスタックトップと呼ぶ。

【0050】

またこのスタックポインタとして、専用の SP 以外に、汎用のレジスタ R_m（レジスタ m）を使うこともできる。このような実装は汎用 CPU を Java（R）仮想マシンとして使用する場合等で、特に有効である。

【0051】

以後特に混乱が起こらない限り、「スタックポインタ（SP）」または「スタックポインタ（SP）アクセス」等の用語を、専用の SP レジスタを使う場合と、汎用のレジスタ（例えば R_m や R_n と表記する）を使用する場合を特に区別せず、両方を含む意味で使って説明する。

【0052】

演算ユニットはデータ移動命令の一部として、SP の内容をマイナス 4 してそのスタックトップにデータを 4 バイトライト（MPU 113 からメモリへ書き込み）する push 命令と、スタックトップからデータを例えば 4 バイトリード（メモリから MPU 113 へ読み出し）して SP の内容をプラス 4 する pop 命令

と、スタックトップからの相対アドレスでメモリ読み出しを行う `load` 命令と、スタックトップからの相対アドレスでメモリ書き込みを行う `store` 命令と、その他直接アドレス、レジスタ間接アドレス、レジスタ相対アドレス等、各種の方法で指定する任意のアドレスに直接ランダムにメモリアクセスする命令と、その他の命令を備えている。

【0053】

図2 (a)、(b)、(c)、(d) の左側「命令」欄に、この `push`、`pop`、`load`、`store` 命令の各命令形式を示す。これらは汎用レジスタ `Rm` または汎用レジスタ `Rn` をスタックポインタとして使用している例である。

【0054】

また、この説明では、これらの命令によるメモリアクセスの単位を4バイトとし、`push/pop` 命令でデータ移動の前後でスタックポインタの増減を行う単位値を4としたが、これは1例であって、どのような値であっても構わない。

【0055】

図2 (a) 左側の「命令」欄は `push` 命令の命令形式を表している。この形式では `Rm` レジスタをスタックポインタとして使用している。図2 (a) 右側の「動作」欄にはこの命令の実行によって行われる動作が示されている。

【0056】

この `push` 命令が実行されるとまず、`Rm` の内容がマイナス4される。

【0057】

次に `Rn` レジスタの内容が `Rm` 間接アドレス指定でメモリに書き込まれる。即ち、`Rn` レジスタの内容が `Rm` レジスタの内容が示すアドレスのメモリに書き込まれる。

【0058】

そして `push/pop` 修飾ビットがオンされる。

【0059】

この `push/pop` 修飾ビットの内容は、図1に示すように、制御ラインを通じて、MPU113からキャッシュユニット117に `push/pop` 命令が実行されたことを知らせるために使われる。

【0060】

図2 (b) 左側の「命令」欄は `pop` 命令の命令形式を表している。この形式では `Rn` レジスタをスタックポインタとして使用している。図2 (b) 右側の「動作」欄にはこの命令の実行によって行われる動作が示されている。

【0061】

この `pop` 命令が実行されるとまず、`push/pop` 修飾ビットがオンされる。

【0062】

`push/pop` 修飾ビットの内容が、制御ラインを通じて、MPU113 からキャッシュユニット117に `push/pop` 命令が実行されたことを知らせるために使われることは前記と同様である。

【0063】

次に、`Rn` 間接アドレス指定でメモリの内容が `Rm` レジスタに読み込まれる。即ち、`Rn` レジスタの内容が示すアドレスのメモリの内容が `Rm` レジスタに読み込まれる。

【0064】

そして、`Rn` の内容がプラス4される。

【0065】

図2 (c) 左側の「命令」欄は `load` 命令の命令形式を表している。この形式では `Rn` レジスタをスタックポインタとして使用している。図2 (c) 右側の「動作」欄にはこの命令の実行によって行われる動作が示されている。

【0066】

この `load` 命令が実行されると、まず `SP` 相対修飾ビットがオンされる。

【0067】

この `SP` 相対修飾ビットの内容は、図1に示すように、制御ラインを通じて、MPU113 からキャッシュユニット117に `SP` 相対アドレス指定で `load/storepop` 命令が実行されたことを知らせるために使われる。

【0068】

次に、`Rn` レジスタの内容にオフセット値を加えた間接アドレス指定でメモリ

の内容がRmレジスタに読み込まれる。即ち、Rnレジスタの内容にオフセット値を加えた値が示すアドレスのメモリの内容がRmレジスタに読み込まれる。

【0069】

SPとして使用されているRnの内容がプラスマイナスされることはない。

【0070】

図2(d)左側の「命令」欄はstore命令の命令形式を表している。この形式ではRmレジスタをスタックポインタとして使用している。図2(d)右側の「動作」欄にはこの命令の実行によって行われる動作が示されている。

【0071】

このstore命令が実行されると、Rnレジスタの内容がRmレジスタの内容にオフセット値を加えた間接アドレス指定でメモリに書き込まれる。即ち、Rnレジスタの内容がRmレジスタの内容にオフセット値を加えた値が示すアドレスのメモリに書き込まれる。

【0072】

そしてSP相対修飾ビットがオンされる。

【0073】

SP相対修飾ビットの内容が、制御ラインを通じて、MPU113からキャッシュユニット117にSP相対アドレス指定でload/storepop命令が実行されたことを知らせるために使われることは前記と同様である。

【0074】

SPとして使用されているRnの内容がプラスマイナスされることはないことも前記と同様である。

【0075】

これらの命令を含むデータ移動命令がMPU113によって実行されると、そのデータ移動メモリが外部メモリにアクセスするものであるならば、MPU113はアクセス対象である外部メモリの論理アドレスを仮想アドレスバスに出力し、データアクセスがデータの読み出しであるか書き込みであるかに応じてキャッシュリード要求またはキャッシュライト要求ビット信号線をそれぞれオンにする。

【0076】

データアクセスがデータのメモリへの書き込みであるならば同時に書き込み対象となるデータを32ビット幅のデータバスにも出力する。

【0077】

仮想データバスに出力された論理アドレスはMMU115によって物理アドレスに変換され、実アドレスバスに出力される。

【0078】

これらのデータを受け取ったキャッシュユニット117は、データアクセスの対象であるメモリがキャッシュ上に存在するか否かを判定する。

【0079】

以上説明したように本実施の形態で使用するMPUからメモリアクセスするために使用する制御信号線は、push/pop修飾、SP相対修飾、キャッシュリード要求、キャッシュライト要求の4本（4ビット）である。MPUによって実行される命令とその動作と各制御線の状態との取り得る組み合わせを図3に示す。

【0080】

図3（a）はキャッシュリード要求とキャッシュライト要求がどちらもオフ（0）であり、push/pop修飾、SP相対修飾の如何に関わらず、キャッシュメモリのリードライトに関わる何等の動作も行われない。

【0081】

図3（b）はキャッシュリード要求がオフ（0）、キャッシュライト要求がオン（1）、push/pop修飾とSP相対修飾のどちらもオフ（0）であり、メモリに対するランダムライトの命令が実行された時の動作とその時の制御信号線の状態が示される。

【0082】

図3（c）はキャッシュリード要求がオフ（0）、キャッシュライト要求がオン（1）、push/pop修飾がオフ（0）、SP相対修飾がオン（1）であり、メモリに対するSP相対ライト（store）命令が実行された時の動作とその時の制御信号線の状態が示される。

【0083】

図3 (d) はキャッシュリード要求がオフ (0)、キャッシュライト要求がオン (1)、push/pop修飾がオン (1)、SP相対修飾がオフ (0) であり、メモリに対するpush命令が実行された時の動作とその時の制御信号線の状態が示される。

【0084】

図3 (e) はキャッシュリード要求がオフ (0)、キャッシュライト要求がオン (1)、push/pop修飾とSP相対修飾のどちらもがオン (1) であり、通常このような状態は想定されておらず、このような状態は起らない。

【0085】

図3 (f) はキャッシュリード要求がオン (1)、キャッシュライト要求がオフ (0)、push/pop修飾とSP相対修飾のどちらもがオフ (0) であり、メモリに対するランダムリードの命令が実行された時の動作とその時の制御信号線の状態が示される。

【0086】

図3 (g) はキャッシュリード要求がオン (1)、キャッシュライト要求がオフ (0)、push/pop修飾がオフ (0)、SP相対修飾がオン (1) であり、メモリに対するSP相対リード (load) 命令が実行された時の動作とその時の制御信号線の状態が示される。

【0087】

図3 (h) はキャッシュリード要求がオン (1)、キャッシュライト要求がオフ (0)、push/pop修飾がオン (1)、SP相対修飾がオフ (0) であり、メモリに対するpop命令が実行された時の動作とその時の制御信号線の状態が示される。

【0088】

図3 (i) はキャッシュリード要求がオン (1)、キャッシュライト要求がオフ (0)、push/pop修飾とSP相対修飾のどちらもがオン (1) であり、通常このような状態は想定されておらず、このような状態は起こらない。

【0089】

図3 (j) はキャッシュリード要求とキャッシュライト要求のどちらかがオン(1)であり、push/pop修飾とSP相対修飾の如何に関わらず、通常このような状態は想定されておらず、このような状態は起こらない。

【0090】

図4は、本実施の形態のキャッシュユニット117のライン構成図であり、各ラインが有するデータ領域と制御に使われる制御ビット(フラグ)が示される。

【0091】

図4に示すように本実施の形態のキャッシュユニット117の各ラインは、16バイト(4ワード、128ビット)からなるデータ領域に加え、20ビットからなるタグ、各1ビットのvalidフラグ、dirtyフラグ、stackフラグを有している。

【0092】

これら3つのフラグの取り得る組み合わせとその時の状態が示す意味を図5に示す。

【0093】

以下にこれらの図を使い、本実施の形態がライトバックモード時に行うキャッシュ制御について説明する。

【0094】

MPU113がランダムリード命令を実行すると、図1に示すMPU113はキャッシュリード要求をオンにし、同時にリードしたいメモリ領域を示す論理アドレスを仮想アドレスバスに出力する。

【0095】

キャッシュユニット117はこれらの信号を入力し、キャッシュビジー応答をオンにする。MPU113はキャッシュビジー応答がオンになったことを確認してキャッシュリード要求をオフにする。

【0096】

MMU115は仮想アドレスバスの上位20ビットを物理アドレスの上位20ビットに変換し、実アドレスバスに出力する。仮想アドレスバスの下位12ビットはそのまま実アドレスバスに出力される。

【0097】

キャッシュユニット117はキャッシュリード要求信号を受け取るとまず、キャッシュビジー応答信号をオンにする。

【0098】

次に実アドレスの中位8ビットをキーとして各ウェイ中の1つのセットを選択し、選択されたセットのタグの値を読み出す。本実施の形態のキャッシュユニットは256セット、4ウェイの構成を仮定しており、その構成を図12に示す。

【0099】

例えばアドレスバスの上位20ビットが2進数表現で「0001 1001 1100 0000 1111 (16進数表現で0x19C0F)」であり、例えばアドレスバスの中位8ビットが2進数表現で「0010 1000 (16進数表現で0x28、10進数表現で『40』)」であったとする。

【0100】

この例の時には、『セット40』に含まれる4つのラインのいずれかのラインがキャッシュに使用する可能性のあるラインとして選択される。

【0101】

しかししながら必ずしもこの構成に限らなければならないものではなく、その他のどのような構成でも本発明を有効に実施することができるが、これについては本発明の本質的な部分と直接関係しないので説明を省略する。

【0102】

次にキャッシュユニット117は実アドレスバスの上位20ビットと読み出したタグの値(20ビット)を比較し、一致したものがあればキャッシュはヒットしたと判断する。

【0103】

例えば上記の例では、アドレスバスの上位20ビットが「0x19C0F」であるので、セット40に含まれるラインに含まれるタグのいずれかの値がこれと同じ「0x19C0F」であるならばキャッシュがヒットしたと判断される。

【0104】

このキャッシュがヒットしたと判断された時の動作を図6 (b) ~ (g) 「動

作」の左欄「ランダムリード」に示す。

【0 1 0 5】

キャッシュがヒットし、且つこのラインの `v a l i d` フラグが 1 の時の動作を図 6 (b) ~ (e) に示す。この時の状態は図 5 (e) ~ (h) に相当し、この時のラインの内容は主記憶装置 1 3 1 の内容と一致しているか主記憶装置 1 3 1 の内容よりも新しいので、このラインに含まれる全 1 6 バイトから実アドレスバスの下位 4 ビットに該当するデータを読み出し、3 2 ビット幅のデータバスに出力し、キャッシュビジー応答信号をオフにする。

【0 1 0 6】

またこの時、`s t a c k` フラグが 1 ならば、図 5 (f)、(h) の状態に相当し、図 6 (c)、(e) に示す通り、`s t a c k` フラグを 0 にする。

【0 1 0 7】

M P U 1 1 3 はキャッシュビジー応答信号がオフになったことを検出して、データバスの内容を読み込む。

【0 1 0 8】

キャッシュはヒットしたが `v a l i d` フラグが 0 の時の動作を図 6 (f)、(g) に示す。この時の状態は図 5 (a)、(b) に相当し、たまたまアドレスバスの上位 2 0 ビットの値とタグの内容は一致したが、既にラインは開放されており、実際にはキャッシュされていなかったことを示している。

【0 1 0 9】

即ちこの状態では、主記憶装置 1 3 1 の内容の方がラインの内容よりも新しいので、対応する主記憶装置 1 3 1 の内容をキャッシュメモリに読み込むリフィルを行うことが必要になる。

【0 1 1 0】

リフィルを行った後、`v a l i d` フラグを 1 にセットし、もしも `s t a c k` フラグが 1 になっていれば 0 にセットする。

【0 1 1 1】

以上の動作によって、図 6 (b) と同じ状態になるので、以後図 6 (b) と同様に、このラインに含まれる全 1 6 バイトから実アドレスバスの下位 4 ビットに

該当するデータを読み出し、32ビット幅のデータバスに出力し、キャッシュビジー応答信号をオフにする。

【0112】

リフィルの詳細については次のキャッシュミスの所で説明する。

【0113】

一方、キャッシュユニット117が実アドレスバスの上位20ビットと読み出したタグの値(20ビット)を比較した結果、選択したセットに含まれるいずれのウエイにも一致したものがなければキャッシュはヒットしなかった(キャッシュミス)と判断する。

【0114】

この時の動作を図6(a)「動作」の左欄「ランダムリード」に示す。

【0115】

まず最初に、同一セット内にある4つのラインから1つを選択し、ラインの開放を行う。

【0116】

開放するラインの選択には例えば、LRUアルゴリズム(Least Recently Used)」を使用する。

【0117】

これを実行するのが図12の各セットに含まれる「LRU式リプレーサ」である。

【0118】

このアルゴリズムは名前の通り、最近、最も使われていないもの(使われる頻度が少なかったもの)を開放する方法であるが、その内容は本発明の本質的部分と直接関係しないので説明を省略する。

【0119】

また必ずしもこのアルゴリズムに限るものではなく、他のアルゴリズムで開放するラインを決定しても勿論構わない。

【0120】

次にこのようにして選択したラインの開放を行う。ラインを開放する動作は各

ラインの状態、即ちその時そのラインの各フラグの値によって示される状態によって異なり、この内容は図7「動作」の右欄に示している。

【0121】

図7(a)の状態ではそのラインが主記憶装置131のキャッシュとして割り付けられていない、即ち開放されていることを示すので、改めで開放する必要はない。この状態は通常 valid フラグが0であることで示される。

【0122】

図7(b)、(c)の状態では、図5(e)、(f)に示すように、ラインの内容と主記憶装置131の内容は一致しているので、ライトバックする必要はなく、只そのラインが開放されたことを示すために valid フラグを0にセットするだけで良い。また図7(c)のように stack フラグが1になっており、stack に割り付けられていたことが示されているならば、そのラインの開放に際し、stack フラグも0にセットする。

【0123】

図7(d)、(e)の状態では、dirty フラグが1になっており、図5(g)、(h)に示すようにそのラインの内容は主記憶装置131の内容よりも新しく更新されているので、ラインの内容を主記憶装置131にライトバックすることが必要になる。

【0124】

このライトバックは図1に示す転送制御ユニット119を使い、転送データバスと外部データバスを介して高速且つ自動的に行われるが、この方法についても本発明の本質と直接関係しないので説明を省略する。

【0125】

ライトバックを行った後、一旦このラインが開放されたことを示すために、valid フラグと dirty フラグを0にセットする。

【0126】

図7(e)に示す valid フラグが1、dirty フラグが1で、stack フラグも1の時は、図5(h)の状態に相当し、このラインがスタックに割り付けられていたことが示されているので、この場合は前記と同様、ラインを開放

するに際しスタックへの割り付けを解除すると共に `s t a c k` フラグも 0 にセットする。

【0127】

図 7 (f)、(g) に示す `v a l i d` フラグが 0 の時は、図 5 (a) ~ (f) の状態に相当し、たまたまタグの値は一致したが、既にこのラインはキャッシュとして使われていない（開放されている）のでライトバックを行う必要はない。もしも `s t a c k` フラグが 1 になっておれば、ラインの開放に際し `s t a c k` フラグを 0 にセットすることだけが行われる。

【0128】

以上で 1 つのラインの開放が終り、次に、この開放したラインをキャッシュとして利用するために、要求されたアドレスを含む 16 バイトの主記憶装置 131 の内容がこのラインにリフィルされる。このリフィルも、図 1 に示す転送制御ユニット 119 を使い、転送データバスと外部データバスを介して高速且つ自動的に行われるが、この方法についても本発明の本質と直接関係しないので説明を省略する。

【0129】

そしてこのラインのタグに実アドレスの上位 20 ビットを書き込み、`v a l i d` フラグを 1、`d i r t y` フラグを 0、`s t a c k` フラグを 0、即ち図 5 (e) の状態にセットする。

【0130】

ランダムリードの実行に際し、この状態は図 6 (b) の状態と同じであるので、以後図 6 (b) と同じ動作が行われる。

【0131】

即ち、このラインに含まれる全 16 バイトから実アドレスバスの下位 4 ビットに該当するデータを読み出し、32 ビット幅のデータバスに出力し、キャッシュビジー応答信号をオフにする。

【0132】

MPU 113 はキャッシュビジー応答信号がオフになったことを見て、データバスの内容を読み込む。

【0133】

MPU113がランダムライト命令を実行すると、図1に示すMPU113はキャッシュライト要求をオンにし、同時にライトしたいメモリ領域を示す論理アドレスを仮想アドレスバスに出力し、更にライトしたいデータを32ビット幅のデータバスに出力する。

【0134】

キャッシュユニット117はこれらの信号を入力し、キャッシュビジー応答をオンにする。MPU113はキャッシュビジー応答がオンになったことを確認してキャッシュライト要求をオフにする。

【0135】

続けてランダムリードと同様の動作によって、セットとウエイの選択と、キャッシュヒットまたはキャッシュミス判断が行われる。これについては既に説明したのでここでは説明を省略する。

【0136】

キャッシュがヒットしたと判断された時の動作を、図6(b)～(g)「動作」の右欄「ランダムライト」に示す。

【0137】

キャッシュがヒットし、且つこのラインのvalidフラグが1の時の動作を図6(b)～(e)に示す。この時の状態は図5(e)～(h)に相当し、この時のラインの内容は主記憶装置131の内容と一致しているか主記憶装置131の内容よりも新しいので、このラインの実アドレスバス下位4ビットに相当する領域にデータバスの内容を書き込み、まだdirtyフラグが1になっていなければ図6(b)、(c)に示すように、dirtyフラグを1にセットし、キャッシュビジー応答信号をオフにする。

【0138】

またこの時、stackフラグが1ならば、図5(f)、(h)の状態に相当し、図6(c)、(e)に示す通り、stackフラグを0にする。

【0139】

MPU113はキャッシュビジー応答信号がオフになったことを見て、ランダ

ムライトの終了したことを知る。

【0140】

キャッシュはヒットしたが `valid` フラグが 0 の時の動作を図 6 (f)、(g) に示す。この時の状態は図 5 (a)、(b) に相当し、たまたまアドレスバスの上位 20 ビットの値とタグの内容は一致したが、既にラインは開放されており、実際にはキャッシュされていなかったことを示している。

【0141】

即ち、主記憶装置 131 の内容の方がラインの内容よりも新しいので、対応する主記憶装置 131 の内容をキャッシュメモリに読み込むリフィルを行うことが必要になる。リフィルを行った後、`valid` フラグを 1 にセットし、このラインの実アドレスバス下位 4 ビットに該当する領域にデータバスの内容を書き込み、`dirty` フラグを 1 にセットする。

【0142】

もしも図 5 (b) の状態に相当し、`stack` フラグが 1 になっていれば、図 6 (g) に示す通り、ラインのスタックへの割り付けを解除すると共に `stack` フラグを 0 にセットする。

【0143】

一方、キャッシュミスと判断された時の動作を図 6 (a) 「動作」の右欄「ランダムライト」に示す。

【0144】

新しくキャッシュに使用するラインを確保するためにアドレスバスの中位 8 ビットで選択したセットから、LRU 式リプレーサが開放するラインを決定して、決定したラインを開放し、開放したラインを新たにキャッシュとして使用するために、リフィルを行う。

【0145】

この動作は既に説明したランダムリードと同じであるからここでは説明を省略する。

【0146】

これらの動作によってラインの状態は主記憶装置 131 の内容と一致し、図 6

(b) の状態になるので、以後図 6 (b) と同じ動作を行う。

【0147】

即ち、このラインの実アドレスバス下位 4 ビットに該当する領域にデータバスの内容を書き込み、キャッシュビジー応答信号をオフにする。

【0148】

MPU113 はキャッシュビジー応答信号がオフになったことを見て、ランダムライトの終了したことを知る。

【0149】

MPU113 が SP 相対リード命令を実行すると、MPU113 はキャッシュリード要求をオンにし、同時にリードしたいメモリ領域を示す論理アドレスを仮想アドレスバスに出力する。

【0150】

同時に MPU113 は SP 相対修飾信号をオンにする。これによってキャッシュユニット 117 は SP 相対リード／ライト命令が実行されたことを知る。この点において、前記ランダムリード実行時と異なる。

【0151】

キャッシュユニット 117 はこれらの信号を入力し、キャッシュビジー応答をオンにする。MPU113 はキャッシュビジー応答がオンになったことを確認してキャッシュリード要求をオフにする。

【0152】

MMU115 は仮想アドレスバスの上位 20 ビットを物理アドレスの上位 20 ビットに変換し、実アドレスバスに出力する。仮想アドレスバスの下位 12 ビットはそのまま実アドレスバスに出力される。

【0153】

キャッシュユニット 117 はキャッシュリード要求信号を受け取るとまず、キャッシュビジー応答信号をオンにする。

【0154】

次に実アドレスの中位 8 ビットをキーとして各ウェイ中の 1 つのセットを選択し、選択されたセットのタグの値を読み出して、キャッシュのヒット、ミスを判

断する。これらの動作はランダムリードと同様である。

【0155】

キャッシュがヒットし、且つこのラインの `valid` フラグが 1 ならばこのラインから実アドレスバスの下位 4 ビットに該当するデータを読み出し、32 ビット幅のデータバスに出力し、キャッシュビジー応答信号をオフにする。

【0156】

この時の動作を図 8 (b) ~ (e) 「動作」の左欄「SP 相対リード」に示す。

【0157】

MPU113 はキャッシュビジー応答信号がオフになったことを見て、データバスの内容を読み込む。

【0158】

キャッシュはヒットしたが `valid` フラグが 0 ならば、前記の通り、たまたまアドレスバスの上位 20 ビットの値とあるラインのタグの内容が一致はしたが、既にそのラインは開放されており、キャッシュされていなかったことを示しているので、リフィルを行った後、`valid` フラグを 1 にセットし、その後図 8 (b) と同じ動作が行われる。この時の動作を図 8 (f)、(g) の左欄「SP 相対リード」に示す。

【0159】

一方、キャッシュユニット 117 が実アドレスバスの上位 20 ビットと読み出したタグの値 (20 ビット) を比較した結果、アドレスバスの中位 8 ビットの値によって選択したセット中のいずれのウェイにも一致したものがなければキャッシュはヒットしなかった (キャッシュミス) と判断し、同一セット内にある 4 つのラインから 1 つを選択してラインの開放を行う。これについても前記ランダムリードと同じであるから説明を省略する。

【0160】

その後、開放したラインに対して該当する主記憶装置 131 からその内容のリフィルを行い、リフィルを行った後 `valid` フラグを 1 にセットし、その後図 8 (b) と同じ動作が行われる。

【0161】

この時の動作を図8（a）の左欄「SP相対リード」に示す。

【0162】

図8（a）～（g）に示すいずれの状態でもstackフラグの設定は行わないので、stackフラグの状態は元の状態のまま維持される。これについてもランダムリードとはその動作を異にしている。

【0163】

MPU113がSP相対ライト命令を実行すると、MPU113はキャッシュライト要求をオンにし、同時にリードしたいメモリ領域を示す論理アドレスを仮想アドレスバスに出力し、更にライトしたいデータを32ビット幅のデータバスに出力する。

【0164】

同時にMPU113はSP相対修飾信号をオンにする。これによってキャッシュユニット117はSP相対リード／ライト命令が実行されたことを知る。この点において、前記ランダムリード実行時とは異なり、前記SP相対リード命令と同じである。

【0165】

キャッシュユニット117はこれらの信号を入力し、キャッシュビジー応答をオンにする。MPU113はキャッシュビジー応答がオンになったことを確認してキャッシュリード要求をオフにする。

【0166】

続けてSP相対リードと同様の動作によって、セットとウェイの選択と、キャッシュヒットまたはキャッシュミスの判断が行われる。これについては前記ランダムリードと同じである。

【0167】

キャッシュがヒットしたと判断された時の動作を、図8（b）～（g）「動作」の右欄「SP相対ライト」に示す。

【0168】

キャッシュがヒットし且つこのラインのvalidフラグが1ならば、このラ

インの実アドレスバス下位 4 ビットに相当する領域にデータバスの内容を書き込み、まだ `dirty` フラグが 1 になっていなければ図 8 (b)、(c) に示すように、`dirty` フラグを 1 にセットし、キャッシュビジー応答信号をオフにする。

【0169】

この時の動作を図 8 (b) ~ (e) 「動作」の右欄「SP 相対ライト」に示す。

【0170】

MPU113 はキャッシュビジー応答信号がオフになったことを見て、SP 相対ライトの終了したことを知る。

【0171】

キャッシュはヒットしたが `valid` フラグが 0 ならば、前記の通りたまたまアドレスバスの上位 20 ビットの値とあるラインのタグの内容が一致はしたが、既にそのラインは開放されており、キャッシュされていなかったことを示しているので、リフィルを行った後、`valid` フラグを 1 にセットし、その後図 8 (b) と同じ動作が行われる。この時の動作を図 8 (f)、(g) の右欄「SP 相対ライト」に示す。

【0172】

一方、キャッシュユニット 117 が実アドレスバスの上位 20 ビットと読み出したタグの値 (20 ビット) を比較した結果、アドレスバスの中位 8 ビットの値によって選択したセット中のいずれのウェイにも一致したものがなければキャッシュはヒットしなかった (キャッシュミス) と判断し、同一セット内にある 4 つのラインから 1 つを選択してラインの開放を行う。

【0173】

その後、開放したラインに対して該当する主記憶装置 131 からその内容のリフィルを行い、リフィルを行った後、`valid` フラグを 1 にセットし、その後図 8 (b) と同じ動作が行われる。この時の動作を図 8 (a) の右欄「SP 相対ライト」に示す。

【0174】

図8 (a) ~ (g) に示すいずれの状態でも `stack` フラグの設定は行わないので、`stack` フラグの状態は元の状態のまま維持される。この点についても前記ランダムライトとは異なり、前記 `SP` 相対リードと同じである。

【0175】

`MPU113` が `pop` 命令を実行すると、`MPU113` はキャッシュリード要求をオンにし、同時に `pop` したいメモリ領域を示す論理アドレスを仮想アドレスバスに出力する。

【0176】

同時に `MPU113` は `push/pop` 修飾信号をオンにする。この点において、前記ランダムリードや `SP` 相対リード実行時と異なる。これによってキャッシュユニット117は `push/pop` 命令が実行されたことを知る。

【0177】

キャッシュユニット117はこれらの信号を入力し、キャッシュビジー応答をオンにする。`MPU113` はキャッシュビジー応答がオンになったことを確認してキャッシュリード要求と `push/pop` 修飾信号をオフにする。

【0178】

`MMU115` は仮想アドレスバスの上位20ビットを物理アドレスの上位20ビットに変換し、実アドレスバスに出力する。仮想アドレスバスの下位12ビットはそのまま実アドレスバスに出力される。

【0179】

キャッシュユニット117はキャッシュリード要求信号を受け取るとまず、キャッシュビジー応答信号をオンにする。

【0180】

次に実アドレスの中位8ビットをキーとして各ウェイ中の1つのセットを選択し、選択されたセットのタグの値を読み出して、キャッシュのヒット、ミスを判断する。これらの動作はランダムリードや `SP` 相対リードと同様であるから説明を省略する。

【0181】

実アドレスバスの上位20ビットの値と選択したセット中の或るラインのタグ

の内容とが一致し、且つこのラインの `valid` フラグが 1 ならばキャッシュはヒットしたと判断し、図 11 (b) ~ (e) の動作を行う。

【0182】

実アドレスバスの第 3 ビット (下の桁から第 4 番目のビット、即ち 2 の 3 乗の桁を表すビット、以後同様) と第 2 ビット (下の桁から第 3 番目のビット、即ち 2 の 2 乗の桁を表すビット、以後同様) の両方が 1 ならば、図 11 (b) ~ (e) の動作左欄「最上位アドレスから `pop`」の動作を行う。

【0183】

即ち、このラインから実アドレスバスの下位 4 ビットに該当するデータを読み出し、32 ビット幅のデータバスに出力し、キャッシュビジー応答信号をオフにする。

【0184】

次に、このラインの `valid` フラグを 0 にし、このラインを開放する。最上位アドレスからデータが `pop` されたことによってこのスタックが空になったためである。

【0185】

この時、`valid` フラグを 0 にセットして、ラインを開放するが、ライトバックは行わない。このラインの内容は再度参照されることはなく、ライトバックして主記憶に保存する必要がないからである。

【0186】

また図 11 (b)、(d) の動作左欄「最上位アドレスから `pop`」に示す通り、このラインの `stack` フラグがオフになっていればオンにする。このラインがスタックとして使用されていたことを記憶しておくためである。

【0187】

また図 11 (d)、(e) の動作左欄「最上位アドレスから `pop`」に示す通り、このラインの `dirty` フラグがオンになっていればオフにする。このラインは開放されるので、`dirty` フラグは意味を持たないからである。

【0188】

実アドレスバスの第 3 ビットと第 2 ビットのいずれか少なくとも一方が 0 なら

ば、図11 (b) ~ (e) の動作右欄「最上位アドレス以外から pop」の動作を行う。

【0189】

即ち、このラインから実アドレスバスの下位4ビットに該当するデータを読み出し、32ビット幅のデータバスに出力し、キャッシュビジー応答信号をオフにする。

【0190】

CPUはキャッシュビジー応答信号がオフになったことを見て、32ビット幅データバスの情報を読み込む。フラグの操作等は行わない。

【0191】

実アドレスバスの上位20ビットの値と選択したセット中の或るラインのタグの内容とが一致したが、このラインの valid フラグが0ならば、図11 (f) (g) の動作を行う。

【0192】

この時は、たまたま選択したセット中の或るラインのタグの内容と実アドレスバスの上位20ビットの値が一致したが、既にこのラインは開放されていたので、リフィルを行う。

【0193】

実アドレスバスの第3ビットと第2ビットの両方が1ならば、図11 (f) (g) の動作左欄「最上位アドレスから pop」の動作を行う。

【0194】

即ち、このラインに対して主記憶装置から該当するアドレスに記憶している情報のリフィルを行い、valid フラグを1にする。そしてこのラインから実アドレスバスの下位4ビットに該当するデータを読み出し、32ビット幅のデータバスに出力し、キャッシュビジー応答信号をオフにする。

【0195】

もしもこの時 stack フラグが0ならば1にする (図11 (f) 動作左欄「最上位アドレスから pop」)。

【0196】

更に、`valid`フラグを0に戻す(図11 (f) (g) 動作左欄「最上位アドレスからpop」)。最上位アドレスからのpopによって、このラインを開放するためである。

【0197】

実アドレスバスの第3ビットと第2ビットの何れか少なくとも一方が0ならば、図11 (f) (g) の動作右欄「最上位アドレス以外からpop」の動作を行う。

【0198】

即ち、このラインに対して主記憶装置から該当するアドレスに記憶している情報のリフィルを実行し、`valid`フラグを1にする。そしてこのラインから実アドレスバスの下位4ビットに該当するデータを読み出し、32ビット幅のデータバスに出力し、キャッシュビジー応答信号をオフにする。フラグの操作等を行わない。

【0199】

実アドレスバスの上位20ビットの値と選択したセット中のラインのタグの内容とが一致するものがなければ、キャッシュはミスしたと判断し、図11 (a) の動作を行う。

【0200】

即ち、このセットからLRUアルゴリズムによって1つのラインを開放し、そのラインに対して主記憶装置から該当するアドレスに記憶している情報のリフィルを実行する。そして`valid`フラグを1に、`dirty`フラグを0にセットし、このラインから実アドレスバスの下位4ビットに該当するデータを読み出し、32ビット幅のデータバスに出力し、キャッシュビジー応答信号をオフにする(図11 (a))。

【0201】

更に、実アドレスバスの第3ビットと第2ビットの両方が1ならば、`stack`フラグを1にする(図11 (a) 動作左欄「最上位アドレスからpop」)。

【0202】

更にまたこの時は、`valid`フラグを0に戻す(図11 (a) 動作左欄「最

上位アドレスから p o p」)。最上位アドレスからの p o p によって、このラインを開放するためである。

【0 2 0 3】

以上の動作の後或いは途中、もしも転送ビジー応答信号がオフならば、転送データバスが空いているので、もうすぐ必要となる可能性のあるメモリについて予めリフィルを実行する。

【0 2 0 4】

具体的には、ラインに含まれるタグの値とそのラインを含むセットによって示される実アドレスの上位 2 8 ビットが、現在の S P の実アドレスの上位 2 8 ビットよりも 1 または 2 大きなラインを 1 つ見付け、図 1 0 の動作右欄「S P が下方から近付いた」に示す動作を行う。

【0 2 0 5】

このようなラインを見付けることができ、且つそのラインの v a l i d フラグが 1 ならば何も行わない（図 1 0 動作右欄（b）～（e））。

【0 2 0 6】

このようなラインを見付けることはできたが、そのラインの v a l i d フラグが 0 であり、且つそのラインがスタックに割り当てられていれば、そのラインのリフィルを行い、v a l i d フラグを 1 にする（図 1 0 動作右欄（g））。

【0 2 0 7】

このようなラインを見付けることはできたが、そのラインの v a l i d フラグが 0 であり、且つそのラインがスタックに割り当てられていなければ、何も行わない（図 1 0 動作右欄（f））。

【0 2 0 8】

またこのようなラインを見付けることができなければ、現在の S P のすぐ上にあり、すぐに p o p によって読み出される可能性の高い主記憶装置の領域がキャッシュされていないことになるので、実アドレスの上位 2 8 ビットが現在の S P の実アドレスの上位 2 8 ビットよりも 1 大きく、下位 4 ビットは 0 である主記憶装置の領域についてキャッシュ上にリフィルを実行する（図 1 0 動作右欄（a））。

【0209】

具体的には、現在のSPの実アドレスの上位28ビットよりも1大きくなアドレスに対応するセットで、LRUによって開放するラインを決定し、決定したラインの開放を行い、そのラインに実アドレスの上位28ビットが現在のSPの実アドレスの上位28ビットよりも1大きく、下位4ビットは0である主記憶装置の領域のリフィルを実行する。そしてそのラインのvalidフラグを1に設定する。

【0210】

また同時に、ラインに含まれるタグの値とそのラインを含むセットによって示される実アドレスの上位28ビットが、現在のSPの実アドレスの上位28ビットよりも小さなラインを捜し、もしもそのようなラインを見付けることができたなら、図7の動作左欄「SPが上方に移動」に示す動作を行う。

【0211】

即ち、このようなラインを見付けることができ、更にそのラインのvalidフラグ、dirtyフラグ、stackフラグがいずれも1であったならば、そのラインのライトバックを行い、dirtyフラグを0にする（図7動作左欄（e））。

【0212】

このようなラインを見付けることができなかったか、或いは見付けることができてvalidフラグ、dirtyフラグ、stackフラグがいずれかが0であったならば何も実行しない（図7動作左欄（a）～（d）（f）（g））。

【0213】

MPU113がpush命令を実行すると、MPU113はキャッシュライト要求信号をオンにし、同時にリードしたいメモリ領域を示す論理アドレスを仮想アドレスバスに出力し、更にライトしたいワードデータを32ビット幅のデータバスに出力する。

【0214】

同時にMPU113はpush/pop修飾信号をオンにする。この点において、前記ランダムリードやSP相対リード実行時とは異なり、前記pop命令時

と同じである。これによってキャッシュユニット117はpush/pop命令が実行されたことを知る。

【0215】

キャッシュユニット117はこれらの信号を入力し、キャッシュビジー応答信号をオンにする。MPU113はキャッシュビジー応答信号がオンになったことを確認してキャッシュライト要求信号とpush/pop修飾信号をオフにする。

【0216】

続けてpopと同様の動作によって、セットとウエイの選択と、キャッシュヒットまたはキャッシュミス判断が行われる。これらの動作はランダムライトやSP相対ライトと同様であるから説明を省略する。

【0217】

実アドレスバスの上位20ビットの値と選択したセット中の或るラインのタグの内容とが一致し、且つこのラインのvalidフラグが1ならばキャッシュはヒットしたと判断し、該当するラインにおいて、図9(b)～(e)の動作左欄「最上位アドレスにpush」または動作右欄「最上位アドレス以外にpush」の動作を行う。

【0218】

実アドレスバスの第3ビットと第2ビットの両方が1ならば、図9(b)～(e)の動作左欄「最上位アドレスにpush」の動作を行う。

【0219】

即ち、32ビット幅のデータバスからライトしたいワードデータを読み込み、このラインの実アドレスバスの下位4ビットに該当する領域にライトし、キャッシュビジー応答信号をオフにする(図9(b)～(e)動作左欄「最上位アドレスにpush」)。

【0220】

次に、もしもこのラインのdirtyフラグが0ならば1にセットする(図9(b)(c)動作左欄「最上位アドレスにpush」)。

【0221】

また、もしもこのラインの `s t a c k` フラグが 0 ならば 1 にセットする（図 9（b）（d）動作左欄「最上位アドレスに `p u s h`」）。

【0222】

CPU はキャッシュビジー応答信号がオフになったことを見て、`p u s h` 命令が完了したことを知る。

【0223】

実アドレスバスの第 3 ビットと第 2 ビットの何れか少なくとも一方が 0 ならば、図 9（b）～（e）の動作右欄「最上位アドレス以外に `p u s h`」の動作を行う。

【0224】

即ち、32 ビット幅のデータバスからライトしたいワードデータを読み込み、このラインの実アドレスバスの下位 4 ビットに該当する領域にライトし、キャッシュビジー応答信号をオフにする（図 9（b）～（e）動作右欄「最上位アドレス以外に `p u s h`」）。

【0225】

次に、もしもこのラインの `d i r t y` フラグが 0 ならば 1 にセットする（図 9（b）（c）動作右欄「最上位アドレス以外に `p u s h`」）。

【0226】

その他のフラグ操作は行わない。

【0227】

CPU はキャッシュビジー応答信号がオフになったことを見て、`p u s h` 命令が完了したことを知る。

【0228】

実アドレスバスの上位 20 ビットの値と選択したセット中の或るラインのタグの内容とが一致したが、このラインの `v a l i d` フラグが 0 ならば、図 9（f）（g）の動作を行う。

【0229】

この時は、たまたま選択したセット中の或るラインのタグの内容と実アドレスバスの上位 20 ビットの値が一致したが、既にこのラインは開放されていたので

、リフィルを行う。

【0230】

実アドレスバスの第3ビットと第2ビットの両方が1ならば、図9 (f) (g) の動作左欄「最上位アドレスにpush」の動作を行う。

【0231】

即ち、このラインに対して主記憶装置から該当するアドレスに記憶している情報のリフィルを行い、validフラグを1にする。そして32ビット幅のデータバスからライトしたいワードデータを読み込み、このラインの実アドレスバスの下位4ビットに該当する領域にライトし、キャッシュビジー応答信号をオフにする(図9 (f) (g) の動作左欄「最上位アドレスにpush」)。CPUはキャッシュビジー応答信号がオフになったことを見て、push命令が完了したことを知る。

【0232】

dirtyフラグは1にセットする(図9 (f) (g) の動作左欄「最上位アドレスにpush」)。

【0233】

もしもこの時stackフラグが0ならば1にする(図9 (f) 動作左欄「最上位アドレスへpush」)。

【0234】

実アドレスバスの第3ビットと第2ビットの何れか少なくとも一方が0ならば、図9 (f) (g) の動作右欄「最上位アドレス以外へpush」の動作を行う。

【0235】

即ち、このラインに対して主記憶装置から該当するアドレスに記憶している情報のリフィルを実行し、validフラグを1にする。そして、32ビット幅のデータバスからライトしたいワードデータを読み込み、このラインの実アドレスバスの下位4ビットに該当する領域にライトし、キャッシュビジー応答信号をオフにする。dirtyフラグは1にセットする。

【0236】

実アドレスバスの上位 2 0 ビットの値と選択したセット中のラインのタグの内容とが一致するものがなければ、キャッシュはミスしたと判断し、図 9 (a) の動作を行う。

【0 2 3 7】

即ち、このセットから L R U アルゴリズムによって 1 つのラインを開放し、そのラインに対して主記憶装置から該当するアドレスに記憶している情報のリフィルを実行する。そして `valid` フラグを 1 に、`dirty` フラグを 0 に、`stack` フラグを 0 にセットする。

【0 2 3 8】

これによって図 9 (b) と同じ状態になるので、以後図 9 (b) の動作を実行する。

【0 2 3 9】

以上の動作の後或いは途中、もしも転送ビジー応答信号がオフならば、転送データバスが空いているので、当分の間必要となる可能性の低いメモリについて予めライトバックを実行する。

【0 2 4 0】

具体的には、MMU のアドレス変換の単位である、ラインに含まれるタグの値の方が現在の S P の実アドレスの上位 2 0 ビットよりも大きいラインか、またはラインに含まれるタグの値とそのラインを含むセットによって示される実アドレスの上位 2 8 ビットが、現在の S P の実アドレスの上位 2 8 ビットよりも 3 以上大きなラインを、1 つ見付け図 1 0 の動作左欄「S P が下方に遠ざかった」に示す動作を行う。

【0 2 4 1】

このようなラインを見付けることはできたが、そのラインの `valid` フラグと `dirty` フラグと `stack` フラグの何れか少なくとも 1 つが 0 ならば何も行わない (図 1 0 動作左欄 (b) ~ (d) (f) (g))。

【0 2 4 2】

このようなラインを見付けることができ、且つそのラインの `valid` フラグと `dirty` フラグと `stack` フラグの全てが 1 ならば、そのラインのライト

バックを行い、dirtyフラグを0にする（図10動作左欄（e））。これによってももしこのラインがLRUアルゴリズムによって開放されることになった時、改めてライトバックを行う必要がなく、処理を高速化することができる。

【0243】

このようなラインを見付けることができなければ、何も行わない（図10動作左欄（a））。

【0244】

（実施の形態2）

本発明の第2の実施の形態であるデータメモリキャッシュ制御装置の作用を、C言語で記述されたプログラムを実行する汎用CPUを例に、図14～図19を使って詳細に説明する。

【0245】

C言語で記述されたプログラムは、例えば上位アドレスから下位アドレス方向に向けて成長するスタック構造を使用する。

【0246】

汎用CPU内に存在する専用スタックポインタ（SP）を使用し、同汎用CPU内に存在する汎用レジスタをフレームポインタ（FP）として使用し、このスタック構造にアクセスを実行する。

【0247】

この時のスタック構造の一例を図13に示す。この例には1つの関数の実行に関わるスタック構造だけを示している。

【0248】

実際には、既に実行された関数コール（呼び出し）の数に対応するスタック構造が、下方に積み重ねられた構造をしている。

【0249】

次に、この構造がC言語による新しい関数呼び出しでどのように生成（変化）されるかを説明する。

【0250】

C言語の関数が呼び出される時にはまず、関数を呼び出す側がスタックトップ

に `push` 命令を使い、関数側での処理に必要な引き数を生成する。この引き数を関数に引き渡すためである。

【0 2 5 1】

この引き数を生成した直後のスタック構造を図 1 4 に示す。

【0 2 5 2】

次に、関数呼び出しを実行する。この時、`push` 命令を実行して関数からの戻りアドレスをスタックに退避する。この関数呼び出し実行直後のスタック構造を図 1 5 に示す。

【0 2 5 3】

次にプログラムの実行は、呼び出された関数側に移行する。

【0 2 5 4】

呼び出された関数側では最初に、`push` 命令を実行し、フレームポインタをも含めた関数内部で使用するレジスタをスタックに退避する。このレジスタ退避直後のスタック構造を図 1 6 に示す。

【0 2 5 5】

次に呼び出された関数では、スタックポインタの値をフレームポインタに設定し、スタックフレームを生成するために、スタックポインタからスタックフレームのサイズを減算する。これによってスタックポインタはより下位アドレス側に移動し、このスタックポインタとフレームポインタの間にスタックフレームが生成される。

【0 2 5 6】

スタックフレームが生成された直後のスタック構造を図 1 7 に示す。

【0 2 5 7】

呼び出された関数側では、次に、フレームポインタ相対リード・ライト命令、スタックポインタ相対リード・ライト命令、ランダムリード・ライト命令等を実行して、引き数にアクセスしたり、スタックフレームにアクセスし、必要な情報にアクセスして関数の処理を実行する。

【0 2 5 8】

また必要なら、他の関数呼び出しを実行することもある。

【0259】

呼び出された関数側では、関数の処理を終了すると、使用していたスタックフレームを破棄するため、スタックポインタにスタックフレームのサイズを加算する。

【0260】

そして `pop` 命令を実行し、退避していたレジスタを復元する。

【0261】

この時のスタック構造は、関数呼び出し直後のスタック構造と完全に同じ状態に復元され、図15に示される。

【0262】

次に `pop` 命令を実行して、呼び出し側の関数へ戻るためのアドレスを取り出し、リターン命令によって、呼び出された関数から呼び出した関数にプログラム実行を復帰させる。

【0263】

この時のスタック構造は、関数に引き渡すための引き数を生成した直後のスタック構造と完全に同一となり、図14に示す。

【0264】

復帰した呼び出し関数側では、引き数列を廃棄するため、スタックポインタに引き数列の大きさを加算する。例えば図14の例では2掛けるワードサイズに相当する値である。

【0265】

以上の処理によって、関数呼び出し処理を開始する前のスタック構造に復帰する。この時のスタック構造は図13に示す通りである。

【0266】

以上の関数呼び出しに関わる一連の処理における、スタックフレーム生成直後のキャッシュラインの状態を図18に示す。

【0267】

図18に示す通り、スタックフレーム以外の部分はスタックとして使用され、そのためにラインの `stack` フラグは1が設定される。

【0268】

キャッシュラインの単位は16バイトであり、スタック構造の単位は4バイトであるので、スタックフレームとレジスタ退避スタックとの境界がキャッシュラインの境界と一致するとは限らない。

【0269】

本発明によると、レジスタ退避スタックとスタックフレームとが同居しているラインでは、`s t a c k`フラグが1に設定される。

【0270】

またスタックフレームと引き数列が同居しているラインでは`s t a c k`フラグが0に設定される。

【0271】

関数処理実行中に、引き数列やスタックフレームに対するアクセスを実行すると、通常これらのアクセスはランダムリード・ライト命令によって行われるため、1に設定されていた`s t a c k`フラグも0に変化することが起ってくる。

【0272】

従って、レジスタ退避スタックとスタックフレームとが同居したラインや、引き数列と戻りアドレスが同居しているラインも、`s t a c k`フラグは0になる。

【0273】

この時の各ラインのフラグの状態を図19に示す。

【0274】

(実施の形態3)

本発明の第3の実施の形態であるデータメモリキャッシュ制御装置の作用を、Java (R) 言語で記述されたプログラムを実行する汎用CPUを例に、図20～図24を使って詳細に説明する。

【0275】

C言語で記述されたプログラムは、例えばC言語で記述されたインタープリタによって実行される。

【0276】

このインタープリタは、C 言語で記述されたプログラムを実行するためのプログラムカウンタの他に、J a v a (R) 仮想マシンを実装するために必要な J a v a (R) プログラムカウンタを有している。

【0 2 7 7】

またスタック構造においても、C 言語で記述されたプログラムを実行するためのスタック構造以外に、J a v a (R) 仮想マシンを実装するために必要な J a v a (R) スタック構造を有している。

【0 2 7 8】

J a v a (R) スタック構造も例えば、上位アドレスから下位アドレス方向に向けて成長するスタック構造を使用することができる。

【0 2 7 9】

この J a v a (R) スタック構造では例えば、汎用 C P U 内に存在する汎用レジスタをスタックポインタ (S P) として使用し、この J a v a (R) スタック構造にアクセスを実行する。

【0 2 8 0】

J a v a (R) スタック構造の一例を図 2 0 に示す。

【0 2 8 1】

図 2 0 には、1 つのメソッド (C 言語における関数に相当する) の処理に対応する J a v a (R) スタック構造が示されている。

【0 2 8 2】

実際には、これまでに発生したメソッド呼び出し (コール: c a l l) に相当する数のスタック構造が、下方に積み重ねられた構造をしている。

【0 2 8 3】

次に、このスタック構造が、J a v a (R) 言語による新しいメソッド呼び出しによってどのように生成 (変化) されるかを説明する。

【0 2 8 4】

最初に、新しいメソッドを呼び出すメソッド呼び出し側が、スタックトップに p u s h 命令を実行することによってメソッド内での処理に使用するための引き数を設定する。

【0285】

この引き数列生成直後の J a v a (R) スタック構造を図 21 に示す。

【0286】

この引き数列は、呼び出されるメソッド側での処理で必要となる値を、メソッド呼び出し側から呼び出されるメソッド側に引き渡すために使用される。

【0287】

次にメソッド呼び出しが実行される。

【0288】

メソッドからの戻り J a v a (R) アドレスは、C 言語スタックに退避される。このアドレスの退避は J a v a (R) スタック構造とは直接関係しないので説明を省略する。

【0289】

このメソッド呼び出しによって、J a v a (R) プログラムの制御は、呼び出されたメソッド側に移行する。

【0290】

呼び出された J a v a (R) メソッド側では、使用するレジスタの退避が必要であったならばこれらのレジスタの値を C 言語スタックに退避する。このレジスタの退避も J a v a (R) スタック構造とは直接関係しないので説明を省略する。

【0291】

次に、呼び出されたメソッドでは、自分自身が実行する処理で必要となるローカル変数のための領域を生成するために、J a v a (R) スタックポインタの値からローカル変数の領域のサイズに相当する値を減算する。

【0292】

このローカル変数領域生成直後の J a v a (R) スタック構造を図 22 に示す。

【0293】

呼び出されたメソッドでは、その後、J a v a (R) スタックポインタ相対アドレス指定である、S P 相対リード・ライト命令によって引き数やローカル変数

にアクセスし、メソッドの実行する。

【0294】

J a v a (R) 言語はスタック指向言語であるため J a v a (R) 命令でのオペランドは、必ず J a v a (R) スタックポインタ相対として指定される。

【0295】

例えば加算 (a d d) 命令ではオペランドが明示的に指定されることはなく、J a v a (R) スタックのトップとセカンドに自動的に設定される。

【0296】

従って、J a v a (R) インタープリタは a d d 命令を実行するには、p o p 命令を2回実行することによって J a v a (R) スタック中のオペランドスタックから2つの値を読み出し、加算を実行し、その結果を p u s h 命令を実行することによって J a v a (R) スタック中のオペランドスタックのトップに格納する。

【0297】

この処理を実行中の J a v a (R) スタック構造を図23に示す。

【0298】

呼び出されたメソッドでは必要に応じて、更に他のメソッドを呼び出すこともある。

【0299】

呼び出されたメソッドでの処理が終了した時点で、J a v a (R) スタック中のオペランドスタックは空になる。

【0300】

そしてローカル変数領域と引き数列を破棄するため、J a v a (R) スタックポインタにこれの合計サイズに相当する値を加算する。

【0301】

この時の J a v a (R) スタック構造は、J a v a (R) メソッド呼び出し処理を開始する前の J a v a (R) スタック構造に復帰する。この時の J a v a (R) スタック構造は図20に示される。

【0302】

以上の J a v a (R) メソッド呼び出し処理の実行における、処理実行中のキャッシュラインの状態を、図 24 に示す。

【0303】

図 24 に示す通り、このメソッド呼び出し処理のために生成したラインの s t a c k フラグは 1 に設定される。

【0304】

また、J a v a (R) スタックポインタがスタックボトムから遠ざかることによって、スタックポインタから離れたラインで転送ビジー応答信号がオフの時間中にライトバックが実行され、そのラインの d i r t y フラグが 0 に設定された状態をも図 24 に示す。

【0305】

【発明の効果】

本特許出願に係る請求項 2 に記載の発明は、C P U からの連続したアドレスのメモリ領域に対する連続書き込み時にキャッシュミスが発生した時、次に書き込みが行われるアドレス方向のメモリ領域に対しては主記憶装置からキャッシュメモリへの読み込みを行わないので、高速化の効率を大きく改善することができる。

【0306】

本特許出願に係る請求項 3 に記載の発明は、C P U による連続したアドレスのメモリ領域に対する連続読み込み時にキャッシュミスが発生した時、既に読み込みが行われたアドレス方向のメモリ領域に対してはキャッシュメモリから主記憶装置への書き込みを行わないので、高速化の効率を大きく改善することができる。

【0307】

本特許出願に係る請求項 4 に記載の発明は、C P U からの連続したアドレスのメモリ領域に対する連続書き込み時に、既に書き込みが行われたアドレス方向のメモリ領域で、現在書き込みが行われているメモリ領域とのアドレス距離が一定以上離れたメモリ領域に対して、バスの空き時間を使って、予めキャッシュメモリから主記憶装置への書き込みを行うので、高速化の効率を大きく改善すること

ができる。

【0308】

本特許出願に係る請求項5に記載の発明は、CPUによる連続したアドレスのメモリ領域に対する連続読み込み時に、今後読み込みが行われるアドレス方向のメモリ領域で、現在読み込みが行われているメモリ領域とのアドレス距離が一定以上近いメモリ領域に対して、バスの空き時間を使って、予め主記憶装置からキャッシュメモリへの読み込みを行うので、高速化の効率を大きく改善することができる。

【図面の簡単な説明】

【図1】

本発明の第1の実施の形態であるデータメモリキャッシュ制御装置のブロック構成を示す図

【図2】

push、pop、load、store命令の各命令形式を示す図

【図3】

MPUによって実行される命令とその動作と各制御線の状態との取り得る組み合わせを示す図

【図4】

本発明の第1の実施の形態のキャッシュユニット117のライン構成図

【図5】

validフラグ、dirtyフラグ、stackフラグの取り得る組み合わせとその時の状態が示す意味を示す図

【図6】

ランダムリード・ランダムライト命令実行時の各ラインの状態に対応した動作の内容を示す図

【図7】

ラインの開放・SPが上方に移動を実行時の各ラインの状態に対応した動作の内容を示す図

【図8】

S P 相対リード・S P 相対ライト命令実行時の各ラインの状態に対応した動作の内容を示す図

【図 9】

p u s h 命令実行時の各ラインの状態に対応した動作の内容を示す図

【図 10】

S P が下方に遠ざかった・S P が下方から近付いた実行時の各ラインの状態に対応した動作の内容を示す図

【図 11】

p o p 命令実行時の各ラインの状態に対応した動作の内容を示す図

【図 12】

本発明の第 1 の実施の形態のキャッシュユニットの構成を示す図

【図 13】

C 言語のスタック構造を示す図

【図 14】

引き数生成直後のスタック構造を示す図

【図 15】

関数呼び出し直後のスタック構造を示す図

【図 16】

レジスタ退避直後のスタック構造を示す図

【図 17】

スタックフレーム生成直後のスタック構造を示す図

【図 18】

スタックフレーム生成直後のキャッシュラインの状態を示す図

【図 19】

ランダムアクセス後のキャッシュラインの状態を示す図

【図 20】

J a v a (R) スタック構造を示す図

【図 21】

引き数生成直後の J a v a (R) スタック構造を示す図

【図 2 2】

ローカル変数領域生成直後の J a v a (R) スタック構造を示す図

【図 2 3】

関数処理中の J a v a (R) スタック構造を示す図

【図 2 4】

関数処理中のキャッシュラインの状態を示す図

【図 2 5】

従来のスタックキャッシュ制御装置において p u s h でキャッシュミスが発生した時に 1 単位の内容が主記憶装置からキャッシュメモリに読み込まれる状態を示す図

【図 2 6】

従来のスタックキャッシュ制御装置において p o p でキャッシュミスが発生した時に 1 単位の内容がキャッシュメモリから主記憶装置にライトバックされる状態を示す図

【図 2 7】

低いアドレスのメモリ領域に記憶されたデータが当分の間はポップによって読み出される可能性が非常に低い状態を示す図

【図 2 8】

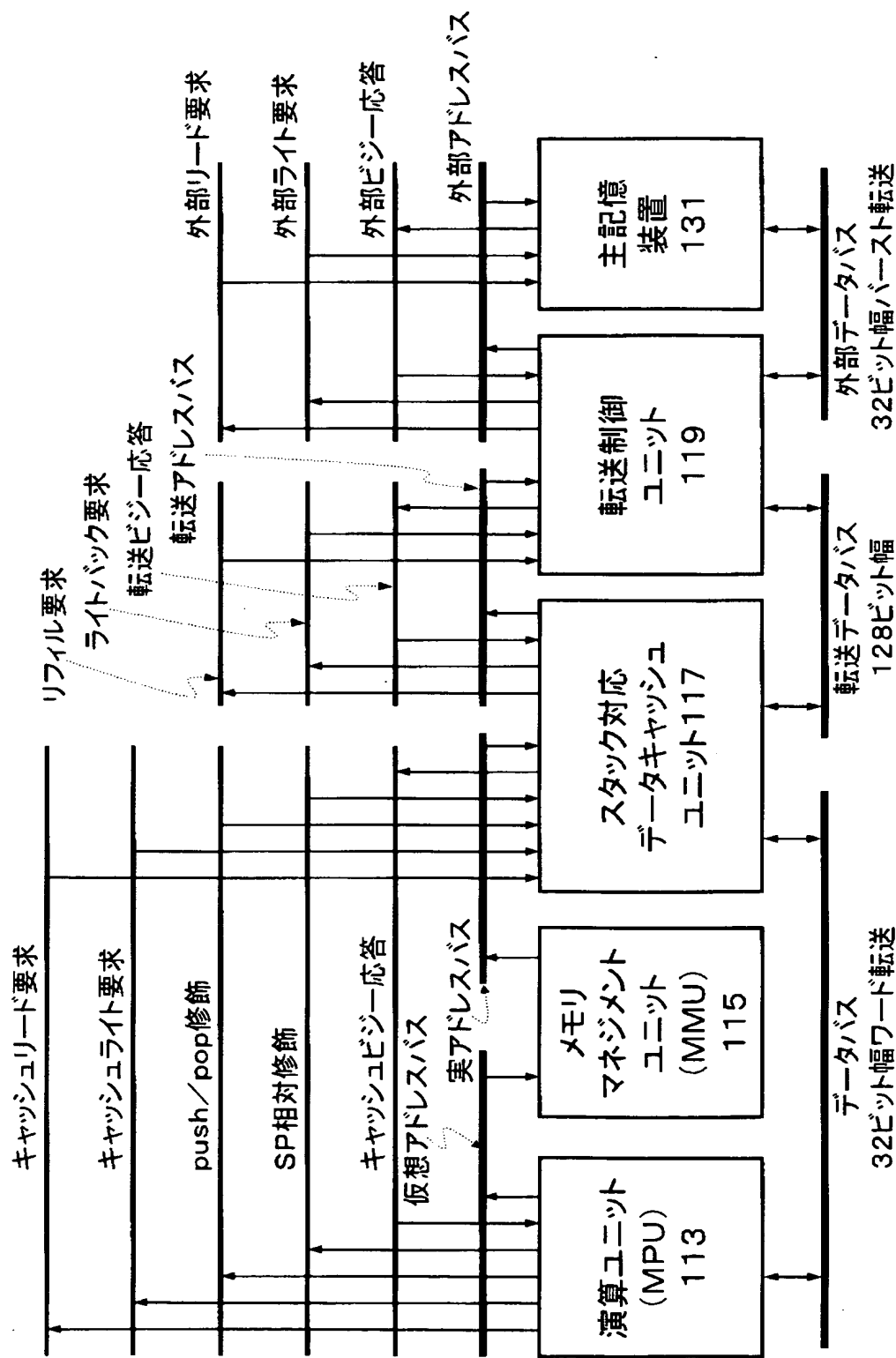
スタックポインタの示すアドレスよりも低い値のアドレス領域はすぐに続けてポップによって読み出される可能性が非常に高い状態を示す図

【符号の説明】

- 1 1 3 演算ユニット (M P U)
- 1 1 5 メモリマネジメントユニット (M M U)
- 1 1 7 スタック対応データキャッシュユニット
- 1 1 9 転送制御ユニット
- 1 3 1 主記憶装置
- 9 1 0 , 9 2 0 , 9 3 0 , 9 4 0 スタックメモリ
- 9 1 3 , 9 2 3 , 9 3 3 , 9 4 3 スタックポインタ

【書類名】
【図 1】

図面



【図 2】

	命 令	動 作
(a)	push Rn, [-Rm]	$Rm = Rm - 4$ $[Rm] = Rn$ push/pop修飾 = 1
(b)	pop [Rn+], Rm	push/pop修飾 = 1 $Rm = [Rn]$ $Rn = Rn + 4$
(c)	load [Rn+オフセット], Rm	SP相対修飾 = 1 $Rm = [Rn + オフセット]$
(d)	store Rn, [Rm+オフセット]	$[Rm + オフセット] = Rn$ SP相対修飾 = 1

【図 3】

	キャッシュ リード要求	キャッシュ ライト要求	push/ pop 修飾	SP 相対 修飾	命令/動作
(a)	0	0	X	X	—
(b)	0	1	0	0	ランダムライト
(c)	0	1	0	1	SP 相対ライト
(d)	0	1	1	0	push
(e)	0	1	1	1	—
(f)	1	0	0	0	ランダムリード
(g)	1	0	0	1	SP 相対リード
(h)	1	0	1	0	pop
(i)	1	0	1	1	—
(j)	1	1	X	X	—

【図 4】

本発明の一実施の形態のキャッシュユニット117のライン構成図

	タグ	validフラグ	dirtyフラグ	stackフラグ	データ領域
ライン1	20ビット	1ビット	1ビット	1ビット	16バイト(4ワード)
ライン2	20ビット	1ビット	1ビット	1ビット	16バイト(4ワード)
ライン3	20ビット	1ビット	1ビット	1ビット	16バイト(4ワード)
ライン4	20ビット	1ビット	1ビット	1ビット	16バイト(4ワード)
:	:	:	:	:	:
:	:	:	:	:	:
:	:	:	:	:	:
:	:	:	:	:	:
:	:	:	:	:	:
:	:	:	:	:	:

【図 5】

	ラインの状態			意味
	validフラグ	dirtyフラグ	stackフラグ	
(a)	0	0	0	tagが指すメモリの内容はラインの内容よりも新しい (現在のこのラインは開放されている)
(b)	0	0	1	ラインはスタックに割り付けられていた tagが指すメモリの内容はラインの内容よりも新しい (現在のこのラインは開放されている)
(c)	0	1	0	—
(d)	0	1	1	—
(e)	1	0	0	tagが指すメモリの内容とラインの内容は一致している
(f)	1	0	1	ラインはスタックに割り付けられている tagが指すメモリの内容とラインの内容は一致している
(g)	1	1	0	ラインの内容はtagが指すメモリの内容よりも新しい
(h)	1	1	1	ラインはスタックに割り付けられている ラインの内容はtagが指すメモリの内容よりも新しい

【図 6】

	ラインの状態			動作	
	validフラグ	dirtyフラグ	stackフラグ	ランダムリード	ランダムライト
(a)		未割り当て		開放するラインの決定 リファイル (b) の動作	開放するラインの決定 決定したラインの開放 リファイル (b) の動作
(b)	1	0	0	データリード	データライト d=1
(c)	1	0	1	データリード s=0	データライト d=1 s=0
(d)	1	1	0	データリード	データライト
(e)	1	1	1	データリード s=0	データライト s=0
(f)	0	0	0	リファイル v=1 データリード	リファイル v=1 データライト d=1
(g)	0	0	1	リファイル v=1 データリード s=0	リファイル v=1 データライト d=1 s=0

【図 7】

	ラインの状態			動作	
	validフラグ	dirtyフラグ	stackフラグ	SPが上方に移動 (SP-tag \geq 4ワード)	ラインの開放
(a)		未割り当て			—
(b)	1	0	0		v=0
(c)	1	0	1		v=0 s=0
(d)	1	1	0		ライトバック v=0 d=0
(e)	1	1	1	ライトバック d=0	ライトバック v=0 d=0 s=0
(f)	0	0	0		
(g)	0	0	1		s=0

【図 8】

	ラインの状態			動作	
	validフラグ	dirtyフラグ	stackフラグ	SP相対リード	SP相対ライト
(a)		未割り当て		開放するラインの決定 リファイル (b) の動作	開放するラインの決定 決定したラインの開放 リファイル (b) の動作
(b)	1	0	0	データリード	データライト d=1
(c)	1	0	1	データリード	データライト d=1
(d)	1	1	0	データリード	データライト
(e)	1	1	1	データリード	データライト
(f)	0	0	0	リファイル v=1 データリード	リファイル v=1 データライト d=1
(g)	0	0	1	リファイル v=1 データリード	リファイル v=1 データライト d=1

【図 9】

	ラインの状態			動作	
	validフラグ	dirtyフラグ	stackフラグ	最上位アドレスにpush	最上位アドレス以外にpush
(a)		未割り当て		開放するラインの決定 決定したラインの開放 リファイル $v=1$ $d=0$ $s=0$ (b) の動作	開放するラインの決定 決定したラインの開放 リファイル $v=1$ $d=0$ $s=0$ (b) の動作
(b)	1	0	0	データライト $d=1$ $s=1$	データライト $d=1$
(c)	1	0	1	データライト $d=1$	データライト $d=1$
(d)	1	1	0	データライト $s=1$	データライト
(e)	1	1	1	データライト	データライト
(f)	0	0	0	リファイル $v=1$ データライト $d=1$ $s=1$	リファイル $v=1$ データライト $d=1$
(g)	0	0	1	リファイル $v=1$ データライト $d=1$	リファイル $v=1$ データライト $d=1$

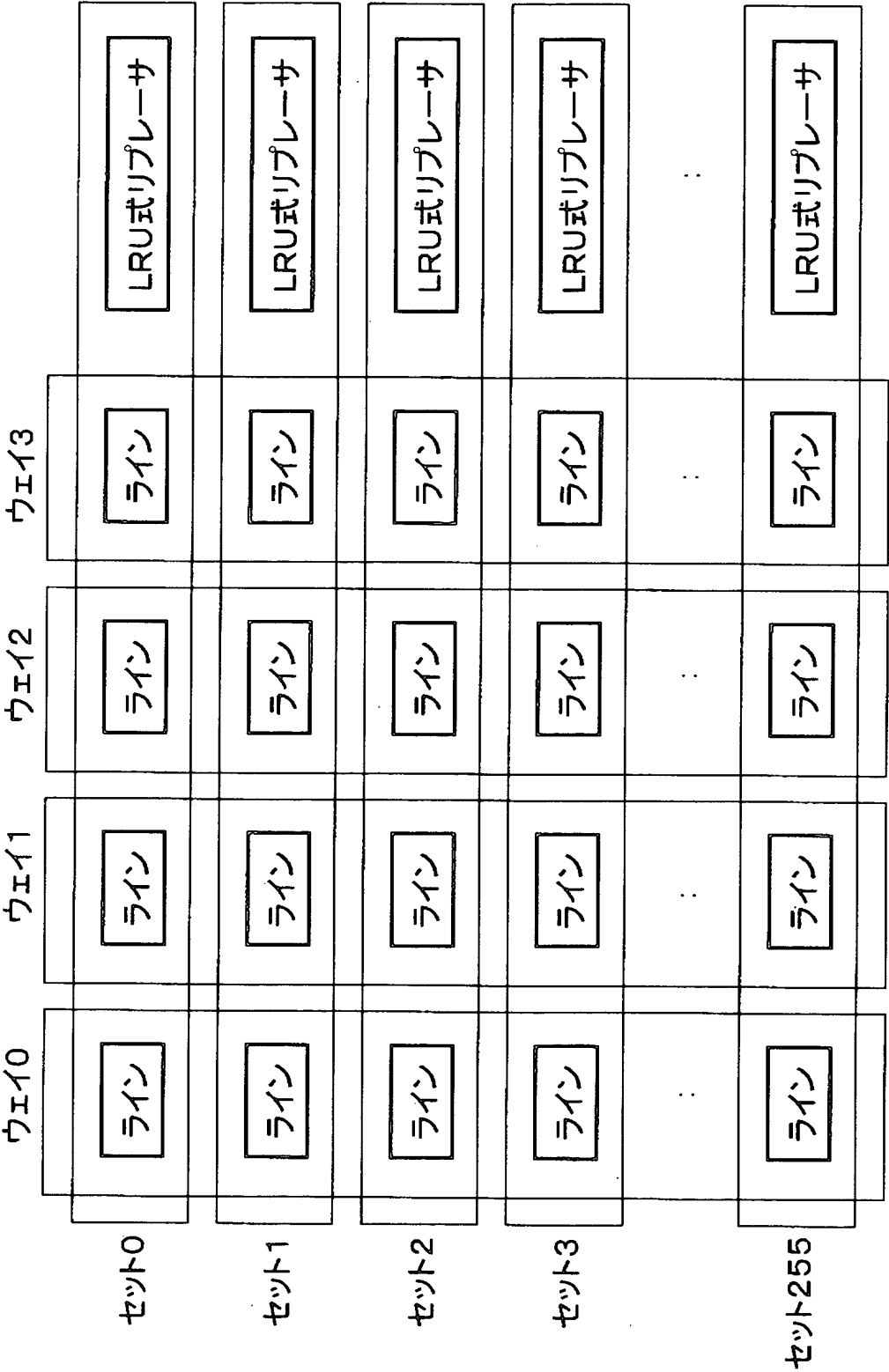
【図 10】

	ラインの状態			動作	
	validフラグ	dirtyフラグ	stackフラグ	SPが下方に遠ざかった (tag-SP \geq 12ワード)	SPが下方から近付いた (tag-SP \leq 8ワード)
(a)	未割り当て				開放するラインの決定 決定したラインの開放 リファイル (g) の動作
(b)	1	0	0		
(c)	1	0	1		
(d)	1	1	0		
(e)	1	1	1	ライトバック d=0	
(f)	0	0	0		
(g)	0	0	1		リファイル v=1

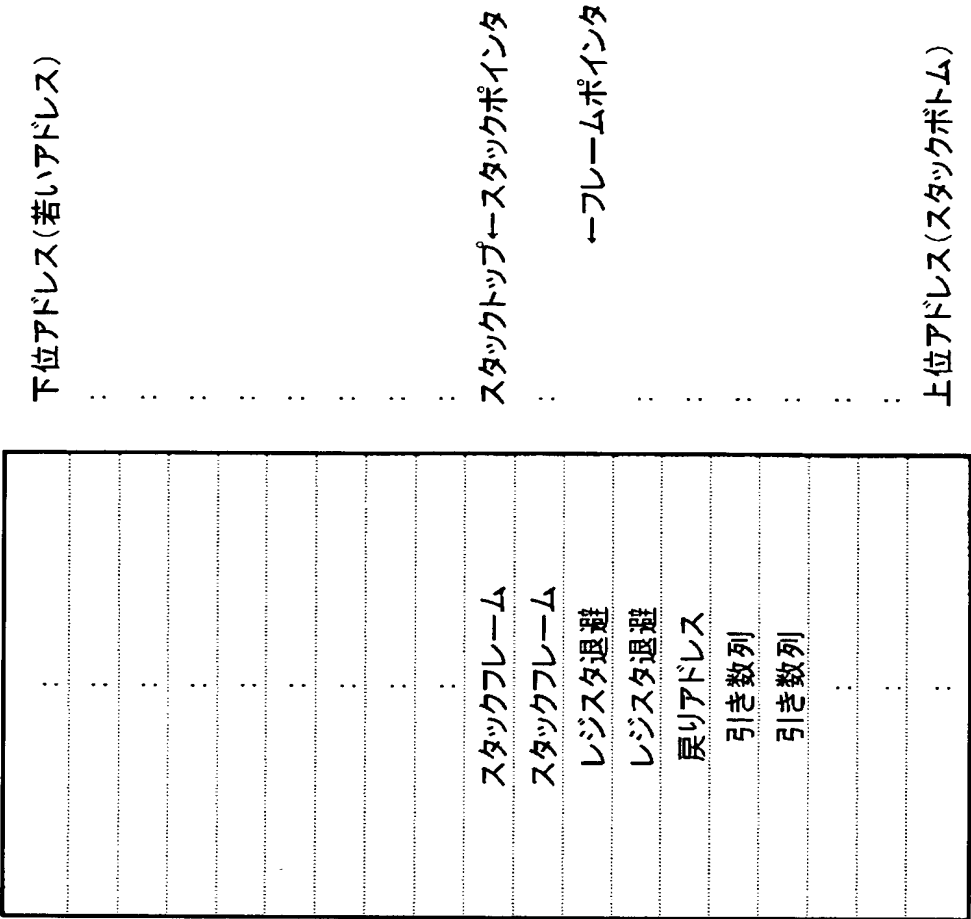
【図 11】

	ラインの状態		動作	
	validフラグ	dirtyフラグ	stackフラグ	最上位アドレスからpop
(a)		未割り当て		開放するラインの決定 決定したラインの開放 リファイル $v=1$ $d=0$ $s=1$ データリード $v=0$ データリード データリード
(b)	1	0	0	データリード $v=0$ $s=1$ データリード
(c)	1	0	1	データリード $v=0$ データリード
(d)	1	1	0	データリード $v=0$ $d=0$ $s=1$ データリード
(e)	1	1	1	データリード $v=0$ $d=0$ データリード
(f)	0	0	0	リファイル $v=1$ データリード $v=0$ $s=1$ データリード
(g)	0	0	1	リファイル $v=1$ データリード $v=0$ データリード

【図 12】

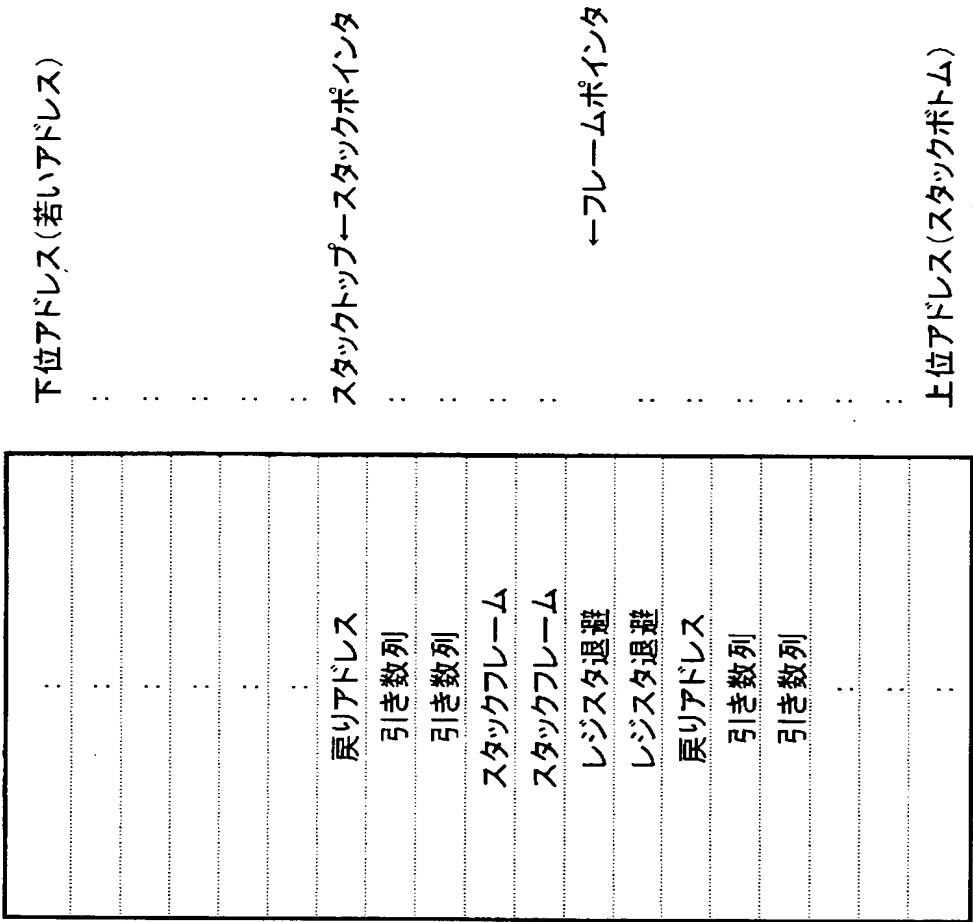


【図 13】

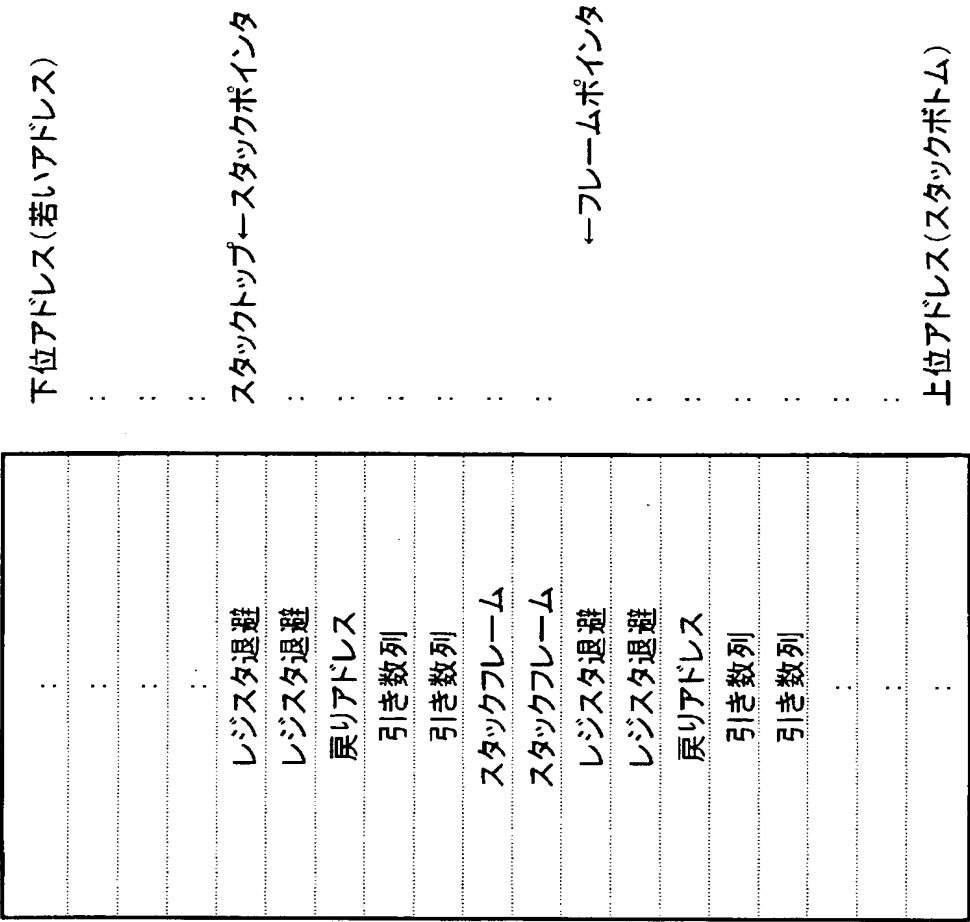


C言語の
スタック構造を示す
図

【図 1 5】

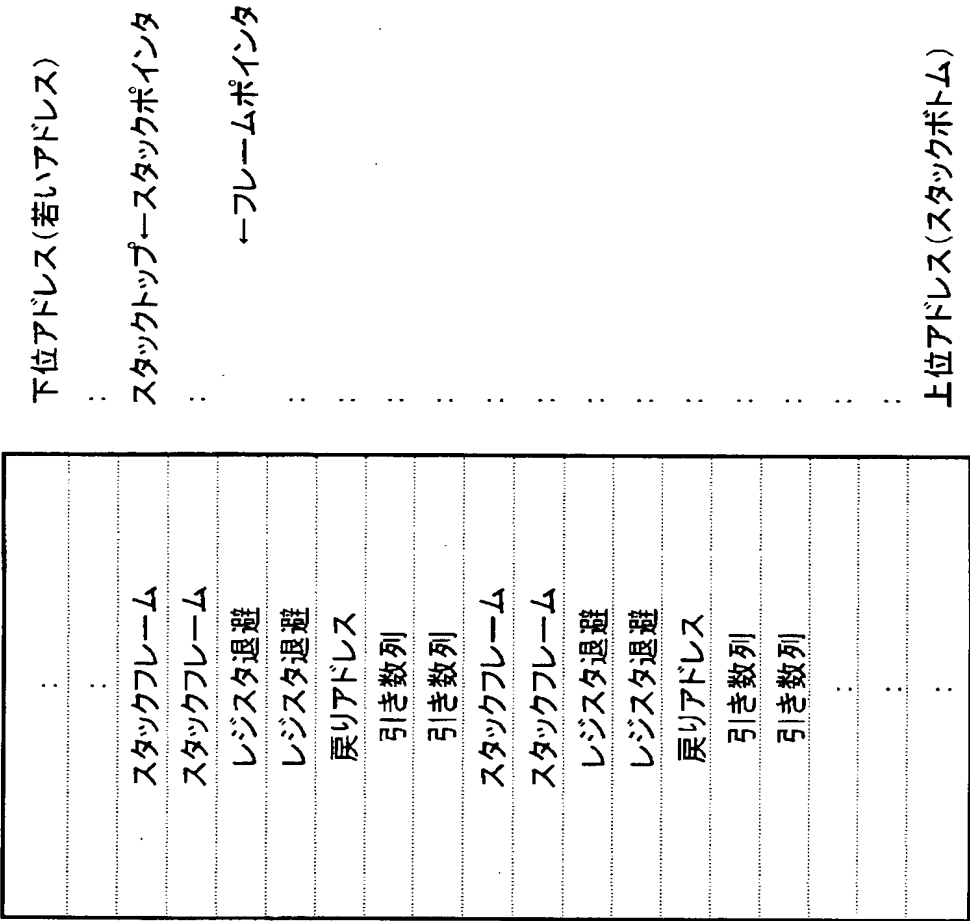


【図 16】



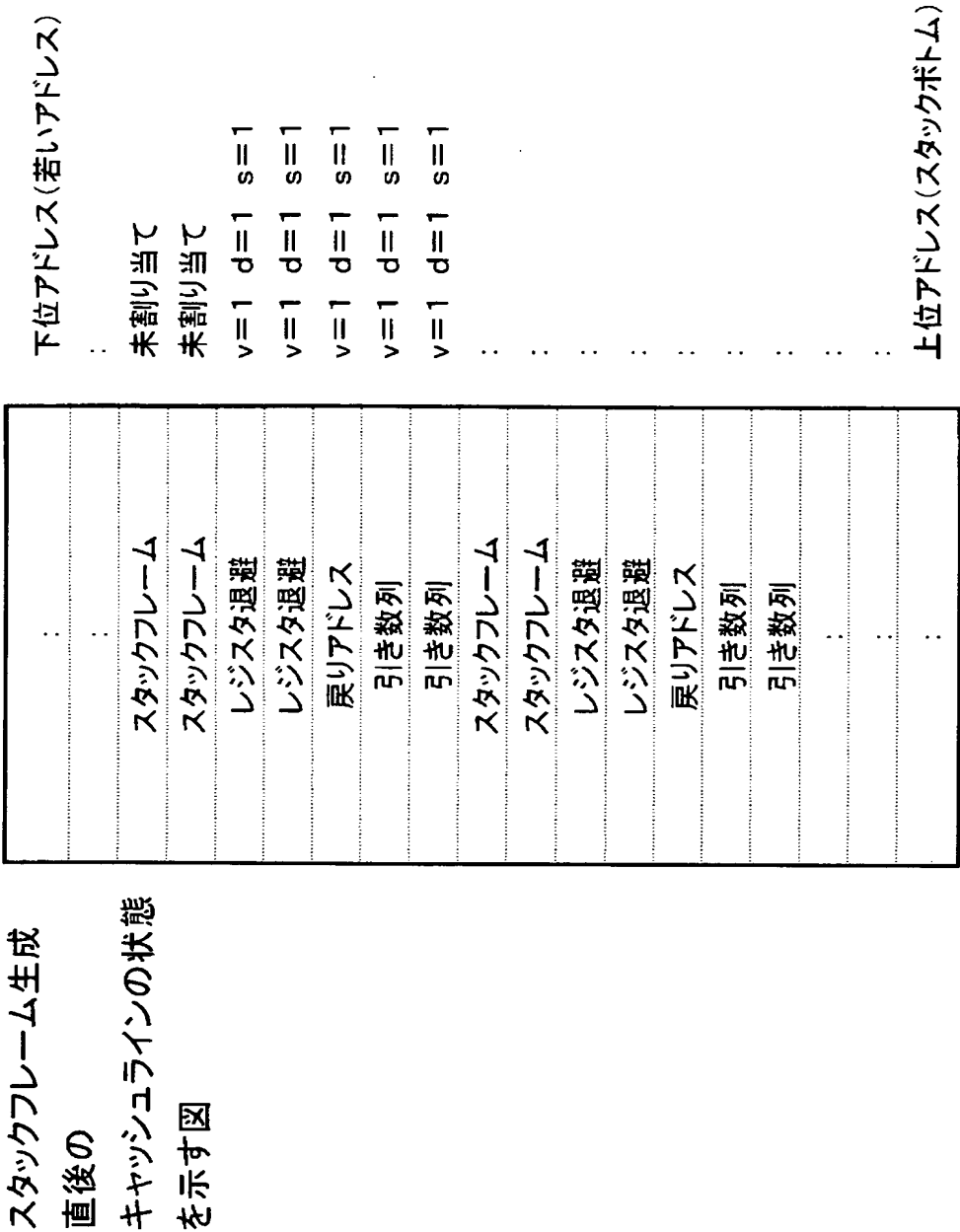
レジスタ退避直後の
スタック構造を示す
図

【図 1 7】

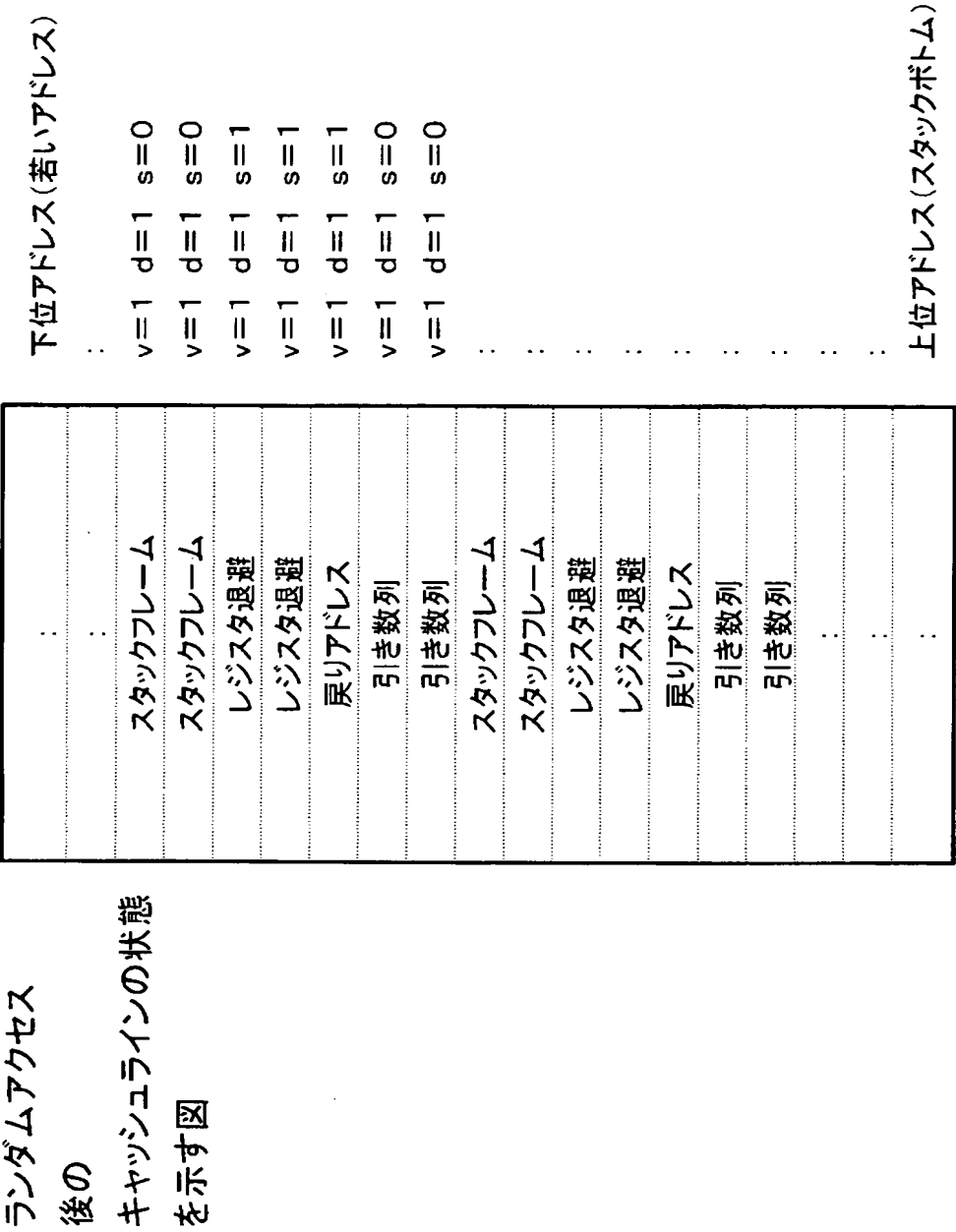


スタックフレーム生成直後のスタック構造を示す
図

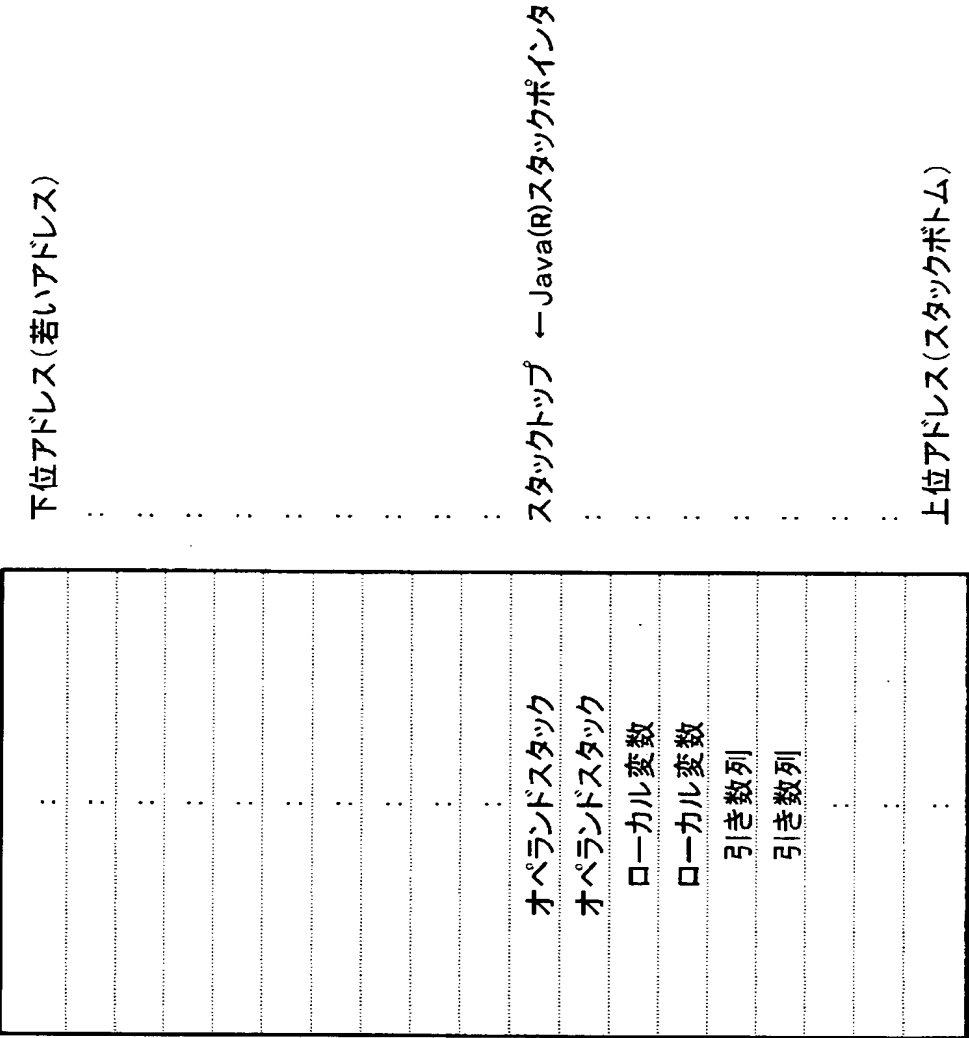
【図 1 8】



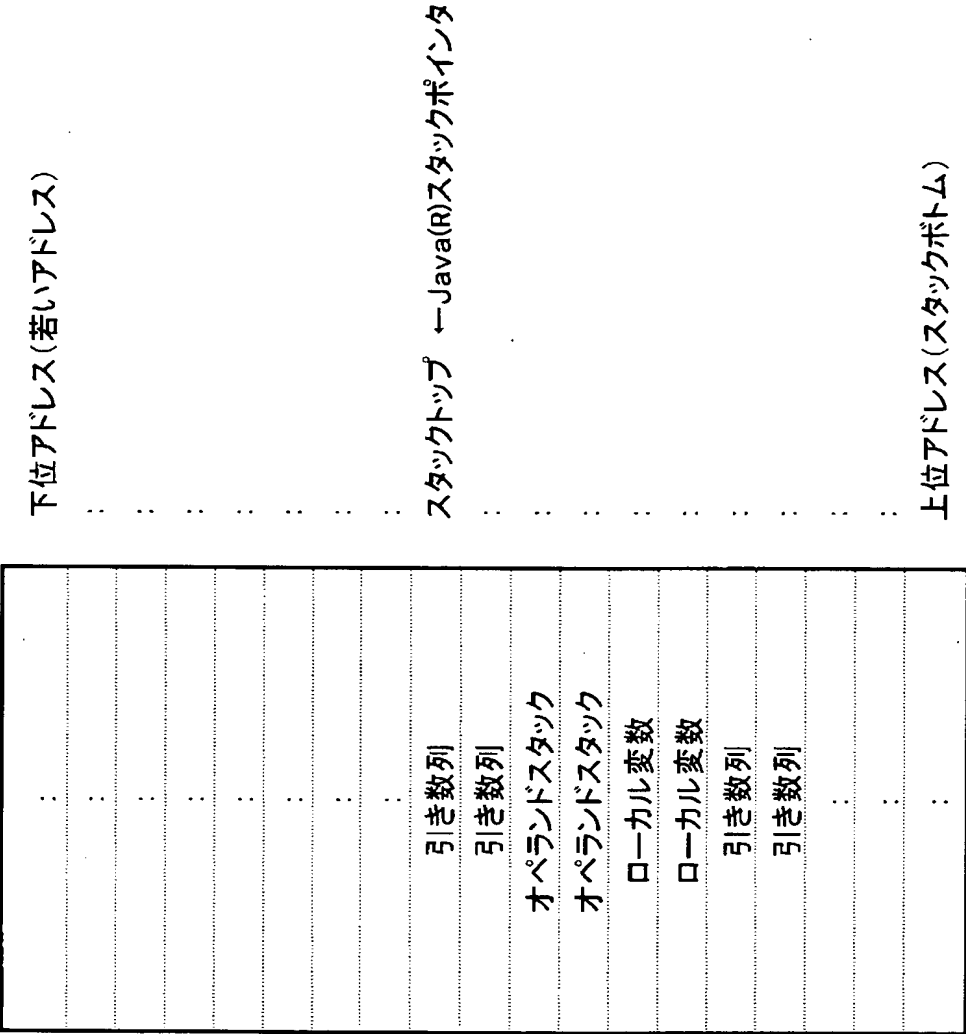
【図 1 9】



【図 2 0】

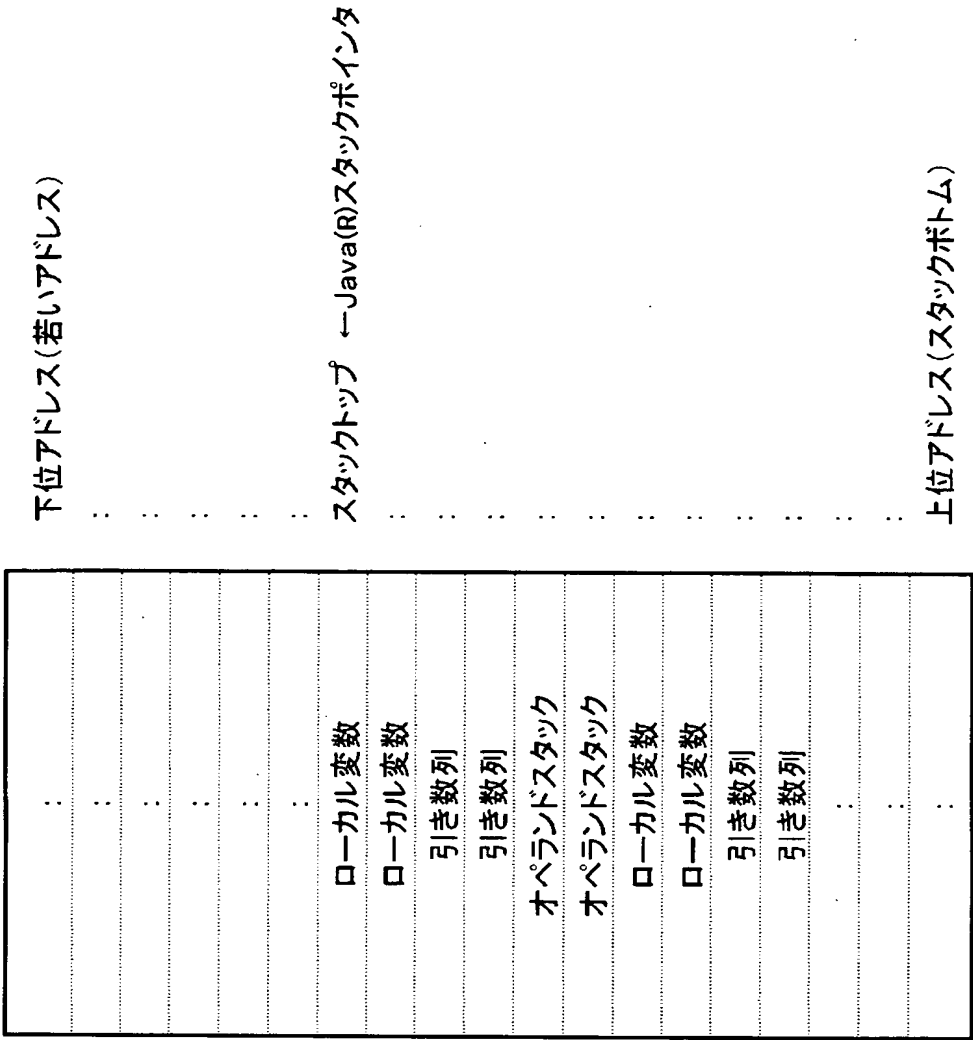


【図 21】



引き数生成直後の
Java(R)スタック構造
を示す図

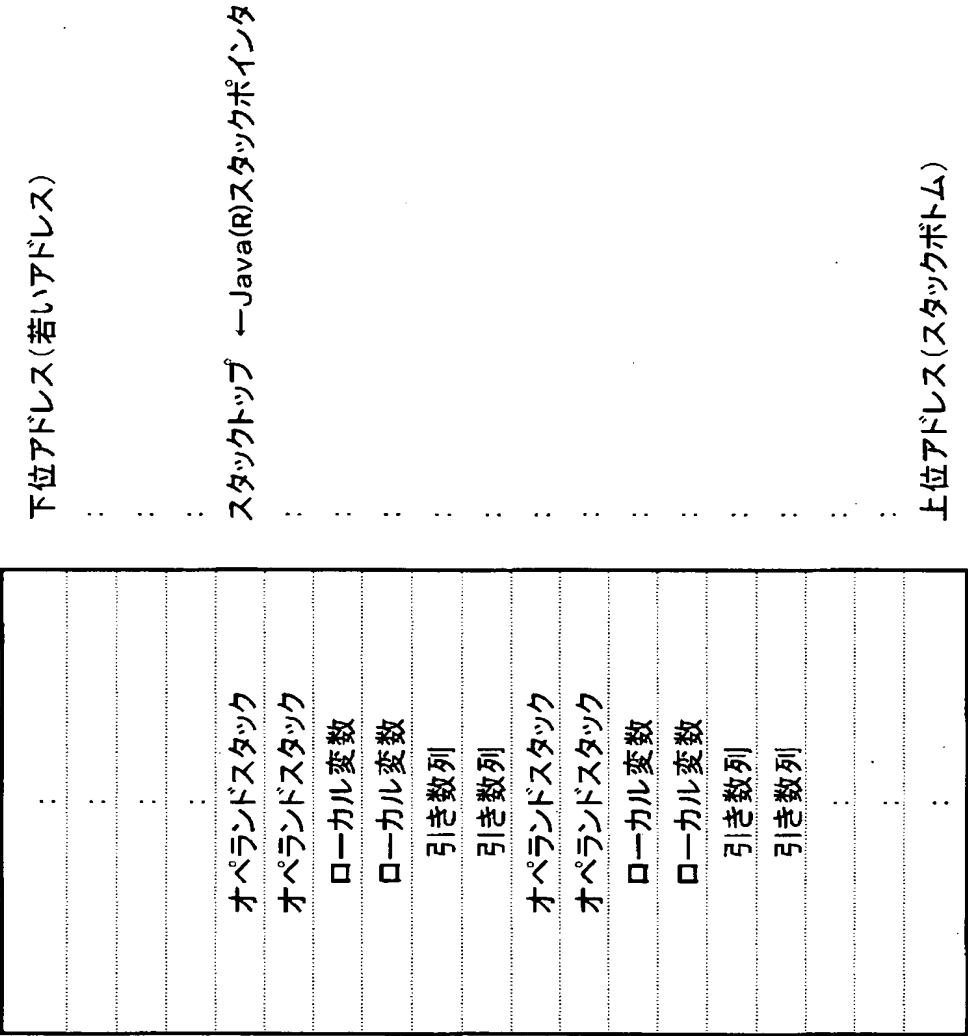
【図 2 2】



ローカル変数領域
生成直後の
Java(R)スタック構造
を示す図

関数処理中の
Java(R)スタック構造
を示す図

【図 2 3】

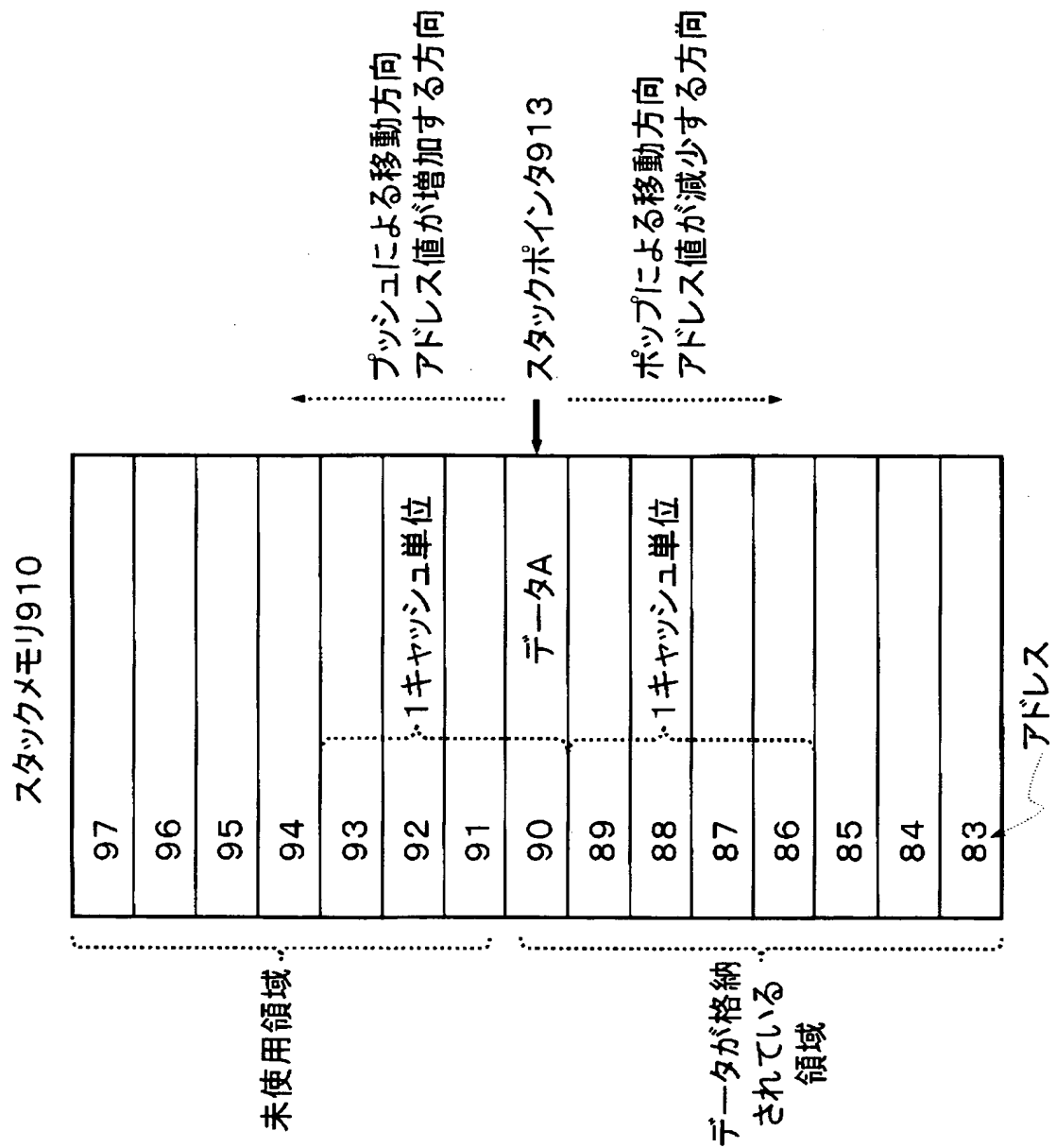


関数処理中の
キャッシュラインの
状態を示す図

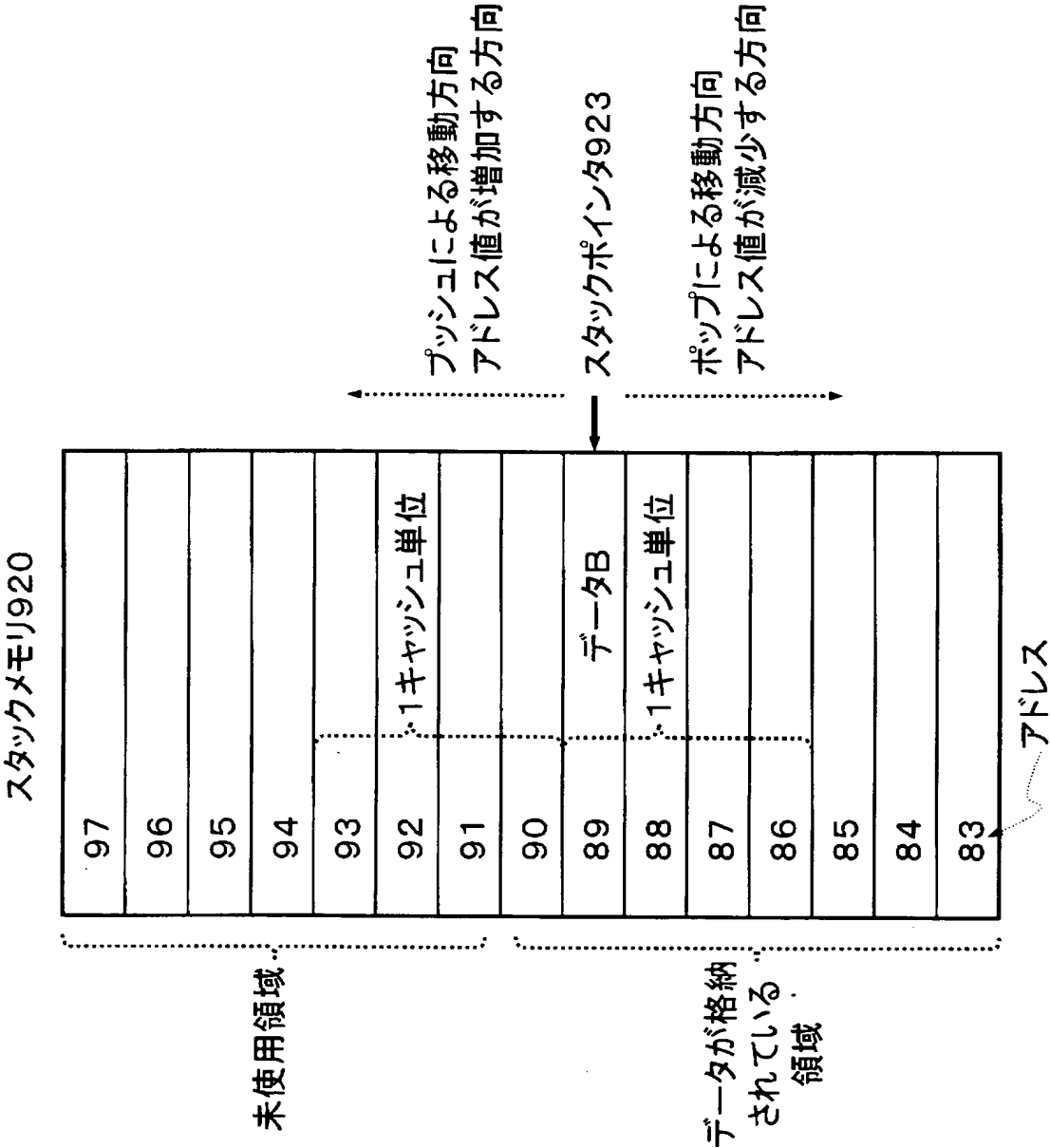
：	下位アドレス(若いアドレス)
：	：
：	：
：	：
オペランドスタック	v=1 d=1 s=1
オペランドスタック	v=1 d=1 s=1
ローカル変数	v=1 d=1 s=1
ローカル変数	v=1 d=0 s=1
引き数列	v=1 d=0 s=1
引き数列	v=1 d=0 s=1
オペランドスタック	：
オペランドスタック	：
ローカル変数	：
ローカル変数	：
引き数列	：
引き数列	：
：	：
：	：
：	上位アドレス(スタックボトム)

【図 2 4】

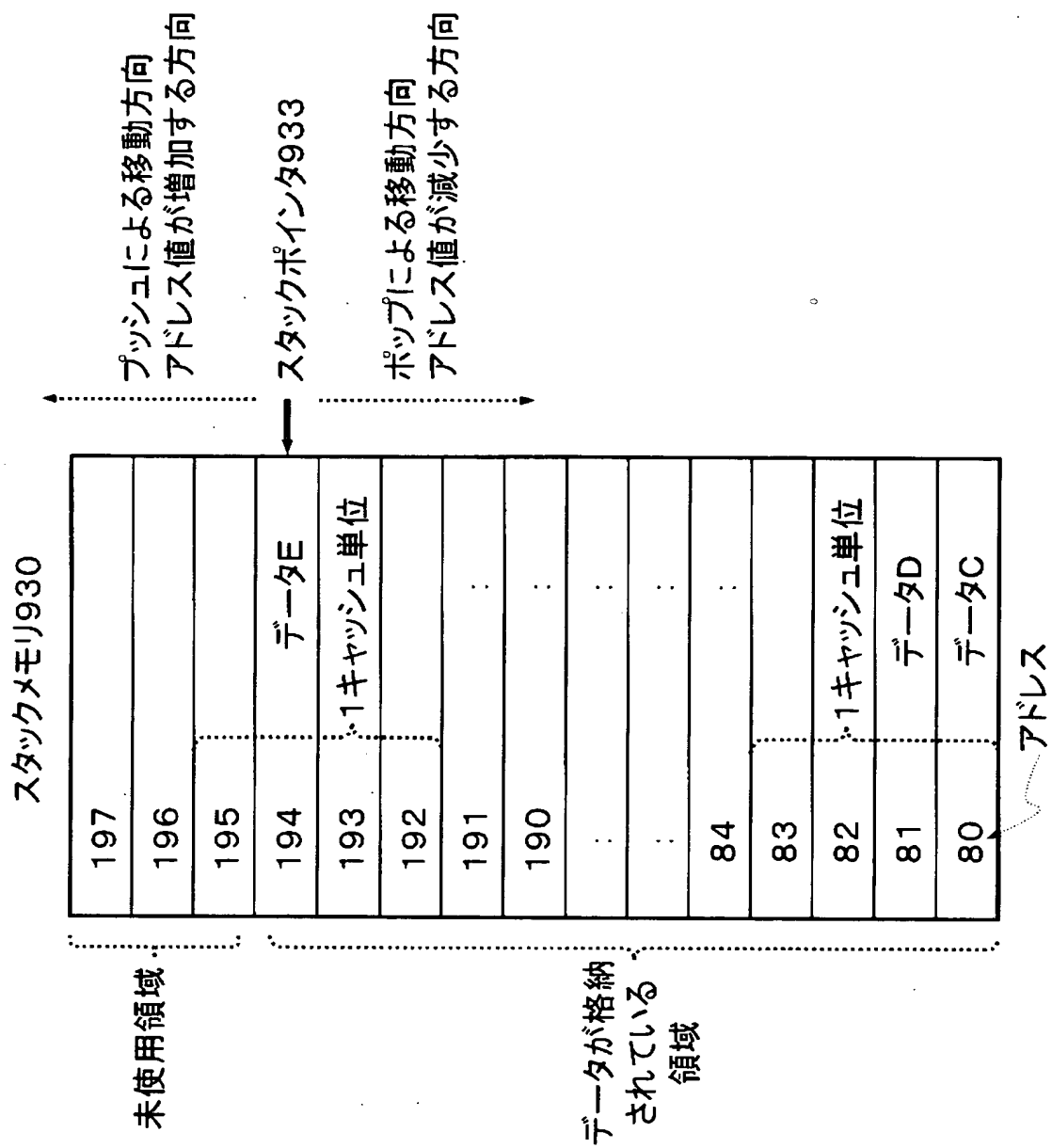
【図 25】



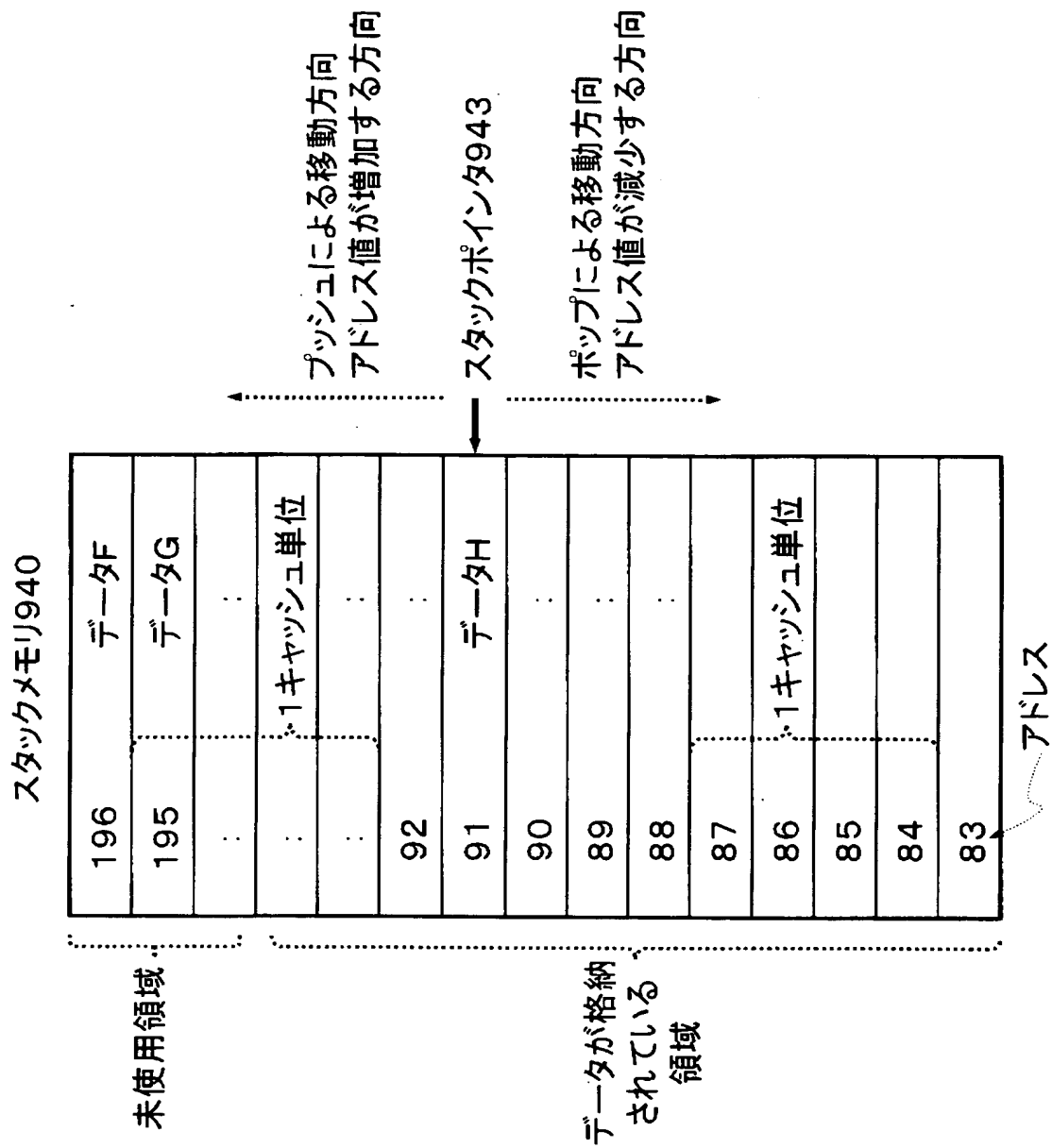
【図 26】



【図 27】



【図 28】



【書類名】 要約書

【要約】

【課題】 スタックに対するメモリアクセスの特性を考慮することで、キャッシュ制御の効率化を行う。

【解決手段】 連続したメモリ領域に対する連続書き込み時にキャッシュミスが発生した時は次に書き込みが行われるメモリ領域に対してリフィルを行わず、連続したメモリ領域に対する連続読み込み時にキャッシュミスが発生した時は既に読み込みが行われたメモリ領域に対してライトバックを行わず、スタックトップから一定以上離れたメモリ領域に対してはバスの空き時間を使って予めライトバックを行い、スタックトップから一定以上近いメモリ領域に対してはバスの空き時間を使って予めリフィルを行う、データキャッシュメモリ制御装置。

【選択図】 図1

特願 2 0 0 3 - 0 7 8 0 2 6

出 願 人 履 歴 情 報

識別番号

[0 0 0 0 0 5 8 2 1]

1. 変更年月日
[変更理由]

1 9 9 0 年 8 月 2 8 日
新規登録

住 所
氏 名

大阪府門真市大字門真 1 0 0 6 番地
松下電器産業株式会社